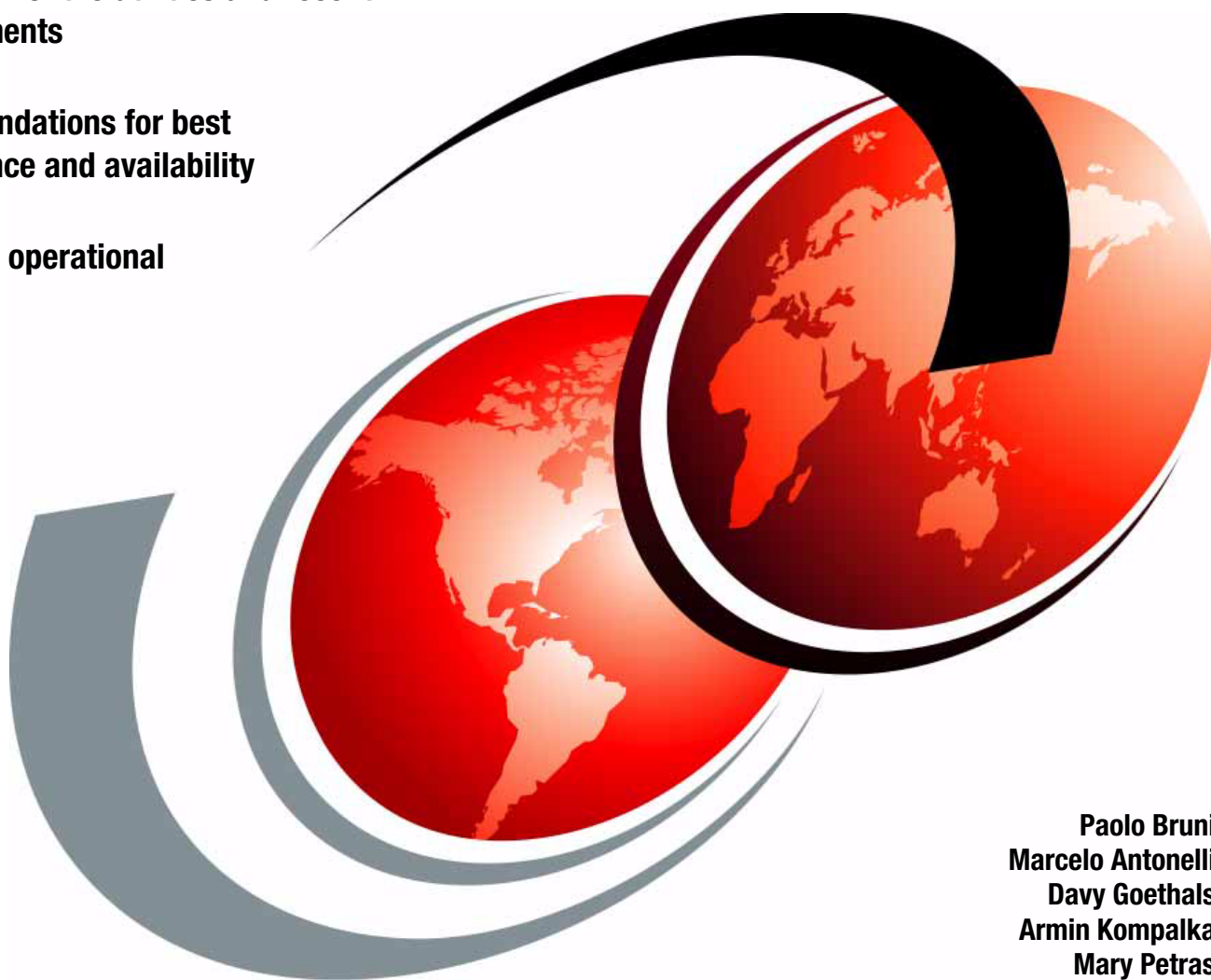


# DB2 9 for z/OS: Using the Utilities Suite

Description of the utilities and recent enhancements

Recommendations for best performance and availability

Advice for operational scenarios



Paolo Bruni  
Marcelo Antonelli  
Davy Goethals  
Armin Kompalka  
Mary Petras

**Redbooks**





International Technical Support Organization

**DB2 9 for z/OS: Using the Utilities Suite**

February 2010

**Note:** Before using this information and the product it supports, read the information in “Notices” on page xxi.

**Second Edition (February 2010)**

This edition applies to Version 9 .1 of IBM DB2 for z/OS (program number 5635-DB2) and the DB2 Version 9 Utilities Suite (program number 5655-N97).

**© Copyright International Business Machines Corporation 2010. All rights reserved.**

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

# Contents

<b>Figures</b> .....	xi
<b>Tables</b> .....	xiii
<b>Examples</b> .....	xv
<b>Notices</b> .....	xxi
Trademarks .....	xxii
<b>Preface</b> .....	xxiii
The team who wrote this book .....	xxiii
Now you can become a published author, too! .....	xxvi
Comments welcome. ....	xxvi
<b>Summary of changes</b> .....	xxvii
February 2010, Second Edition .....	xxvii
April 2010, First Update .....	xxvii
<b>Part 1. Introduction</b> .....	1
<b>Chapter 1. Introduction</b> .....	3
1.1 Contents of this book .....	4
1.2 Brief overview of all DB2 utilities .....	4
1.2.1 Online utilities .....	5
1.2.2 Stand-alone utilities .....	12
1.3 Utilities at work .....	15
1.4 Major changes since DB2 7 .....	19
1.4.1 Packaging of utilities .....	19
1.4.2 Functional enhancements .....	20
<b>Part 2. Planning for DB2 utilities</b> .....	25
<b>Chapter 2. Managing partitions</b> .....	27
2.1 Why partitioning .....	28
2.2 Partitioning, non-partitioning, and data-partitioned secondary indexes and partitioned tables .....	29
2.2.1 Index-controlled partitioning .....	29
2.2.2 Table-controlled partitioning .....	30
2.3 Universal table spaces .....	32
2.3.1 Partition-by-growth table spaces .....	33
2.3.2 Range-partitioned universal table space .....	37
2.4 Partition management .....	39
2.4.1 Adding a partition to an existing regular-partitioned or range-partitioned universal table space .....	39
2.4.2 Rotate partition .....	40
2.4.3 Rebalance partitions dynamically .....	41
2.4.4 ALTER partition boundary .....	42
2.5 SQL and partitioning .....	43
<b>Chapter 3. Simplifying utilities with wildcarding and templates</b> .....	47

3.1 Wildcards . . . . .	48
3.1.1 What is wildcarding . . . . .	48
3.1.2 Benefits of using wildcards . . . . .	48
3.1.3 How to use wildcarding . . . . .	48
3.1.4 Using multiple wildcards . . . . .	50
3.1.5 LISTDEF expansion . . . . .	51
3.1.6 Recommendations for the use of wildcards . . . . .	54
3.1.7 Current restrictions in the use of wildcards . . . . .	55
3.2 Templates . . . . .	55
3.2.1 Benefits of using templates . . . . .	55
3.2.2 How to use templates . . . . .	56
3.2.3 Naming standards . . . . .	57
3.2.4 Substring notation support . . . . .	58
3.2.5 Intelligent data set sizing . . . . .	60
3.2.6 Template switching function . . . . .	60
3.2.7 Template with LOBs and file reference variables . . . . .	62
3.2.8 TEMPLATE with UNIX file (HFS) or pipe . . . . .	64
3.2.9 Recommendations for the use of templates . . . . .	66
3.2.10 Current restrictions in the use of templates . . . . .	66
3.3 Combining wildcards and templates . . . . .	66
3.3.1 Compatibility with utilities . . . . .	68
3.3.2 Mixing the old and the new . . . . .	68
3.3.3 Object in restricted status . . . . .	69
3.3.4 Utility restartability . . . . .	69
3.3.5 Use with partitions . . . . .	71
3.4 OPTIONS . . . . .	72
3.4.1 OPTIONS(PREVIEW) . . . . .	72
3.4.2 Other OPTIONS functionality . . . . .	73
3.5 Using Unicode control statements . . . . .	74
<b>Chapter 4. Performance via parallelism of executions . . . . .</b>	<b>75</b>
4.1 Levels of parallelism . . . . .	76
4.2 Inline executions . . . . .	76
4.2.1 Inline COPY . . . . .	77
4.2.2 Inline RUNSTATS . . . . .	79
4.3 COPY and RECOVER of a list of DB2 objects . . . . .	80
4.3.1 COPY in parallel . . . . .	80
4.3.2 RECOVER in parallel . . . . .	81
4.4 Parallelism with the LOAD utility . . . . .	82
4.5 Partition parallelism with the UNLOAD utility . . . . .	84
4.6 Parallelism with the REORG TABLESPACE utility . . . . .	85
4.7 Parallelism with the REBUILD INDEX utility . . . . .	86
4.7.1 Rebuilding the partitioning index of a partitioned table space . . . . .	86
4.7.2 Rebuilding a non-partitioning index . . . . .	87
4.7.3 Rebuilding all indexes of a partitioned table space . . . . .	88
4.7.4 Rebuilding all indexes of a non-partitioned table space . . . . .	88
4.8 Parallelism with the CHECK INDEX utility . . . . .	89
4.9 Considerations for running parallel subtasks . . . . .	90
<b>Chapter 5. Invoking and controlling executions . . . . .</b>	<b>93</b>
5.1 Triggering utility executions . . . . .	94
5.2 Important DB2 catalog space statistics . . . . .	95
5.2.1 CLUSTERRATIOF and DATAREPEATFACTOR . . . . .	95

5.2.2	NEAROFFPOSF, FAROFFPOSF, and CARDF	97
5.2.3	NEARINDREF and FARINDREF	99
5.2.4	Relationship of *OFFPOS, *INDREF, and CLUSTERRATIOF	100
5.2.5	CHANGELIMIT option	103
5.2.6	AVGROWLEN and AVGKEYLEN	104
5.2.7	LEAFNEAR and LEAFFAR	107
5.2.8	LEAFDIST and LEAFDISTLIMIT	110
5.2.9	Trigger limits	111
5.2.10	VSAM data set extents	112
5.2.11	Reclaiming dead space	112
5.3	RUNSTATS	113
5.3.1	Space statistics collected by RUNSTATS	113
5.3.2	When to run RUNSTATS	115
5.3.3	Frequently asked questions	116
5.4	What is new with real-time statistics (RTS)	117
5.4.1	When DB2 externalizes RTS	118
5.4.2	When are rows added/removed to/from RTS tables	119
5.4.3	Establish a base value for RTS	119
5.4.4	Incremental updates to RTS	121
5.4.5	RTS manager	123
5.4.6	New RTS columns	124
5.4.7	Accuracy of RTS	124
5.4.8	Considerations on RTS usage	125
5.4.9	Recommendations	129
5.5	Trends from historical statistics	134
5.5.1	Monitoring space growth	134
5.5.2	Compression dictionary considerations	135
5.6	Stored procedure DSNACCOX	135
5.7	DB2 Administrative Task Scheduler	136
5.8	IBM DB2 Tools	138
5.8.1	DB2 Administration Tool	139
5.8.2	DB2 Utility Enhancement Tool	139
5.8.3	DB2 Automation Tool	139
5.8.4	DB2 High Performance Unload for z/OS	139
5.8.5	DB2 Recovery Expert for z/OS	139
5.8.6	DB2 Change Accumulation Tool	140
5.8.7	DB2 Storage Management Utility	140
5.8.8	Optim Query Tuner for DB2 for z/OS and Optim Query Workload Tuner for DB2 for z/OS	140
5.8.9	DB2 Buffer Pool Analyzer	140
<b>Chapter 6.</b>	<b>Sort processing</b>	<b>141</b>
6.1	The DFSORT program	142
6.2	Sort data sets used by DB2 utilities	142
6.2.1	Work data sets used outside DFSORT	143
6.2.2	Work data sets used by DFSORT	143
6.3	Best DFSORT default options for DB2	148
6.4	Main memory used by DFSORT	151
6.5	DSNUTILB and DSNUTILS region sizes	154
6.6	DFSORT zIIP offload capability	155
6.7	Debugging DFSORT problems	156
6.8	DB2 allocated sort work data sets	159

<b>Part 3. Executing DB2 utilities</b> .....	165
<b>Chapter 7. Loading and unloading data</b> .....	167
7.1 Best practices to obtain good LOAD and UNLOAD performance .....	170
7.1.1 Invoke parallelism .....	170
7.1.2 Use the REUSE option with LOAD .....	174
7.1.3 Other things that will influence the performance .....	175
7.2 Loading into reordered row format .....	176
7.3 Loading and unloading data in delimited format .....	178
7.4 Loading and unloading compressed data .....	179
7.4.1 KEEPDICTIONARY .....	180
7.4.2 COPYDICTIONARY .....	181
7.4.3 Using SLIDING SCALE .....	181
7.5 Loading and unloading Unicode data .....	182
7.6 Loading and unloading tables with multilevel security .....	183
7.7 NOCOPYPEND option .....	183
7.8 Loading NOT LOGGED table spaces .....	185
7.9 Unloading from a clone table .....	186
7.10 Unloading from an image copy .....	187
7.11 Loading and unloading decimal floating-point data .....	188
7.12 Loading and unloading LOB and XML data .....	189
7.12.1 Loading and unloading LOB or XML data as normal data columns .....	190
7.12.2 Loading and unloading LOB or XML data using file reference variables .....	191
7.12.3 Loading LOB data with the cross loader .....	193
7.12.4 Loading LOB data with SORTKEYS NO .....	194
7.13 Cross loader considerations .....	195
7.14 LOAD SHRLEVEL CHANGE and clone tables .....	198
7.15 Skip locked data during UNLOAD .....	199
7.16 Loading and unloading HFS files .....	200
7.17 Loading and unloading virtual files .....	202
<b>Chapter 8. Reorganizing data</b> .....	205
8.1 Best practices when using the REORG utility .....	206
8.1.1 Use online REORG .....	206
8.1.2 How to get the best REORG performance .....	206
8.1.3 Controlling online REORG .....	212
8.1.4 Other recommendations .....	214
8.2 The elimination of the BUILD2 phase in DB2 9 .....	215
8.3 Online REORG of LOB and XML data .....	217
8.3.1 Online REORG of a LOB table space .....	217
8.3.2 Online REORG of an XML table space .....	219
<b>Chapter 9. Copying data</b> .....	223
9.1 COPY .....	224
9.1.1 Inline COPY with LOAD and REORG .....	224
9.1.2 Copying indexes .....	224
9.1.3 Lists .....	227
9.1.4 COPY parallelism .....	228
9.1.5 The CHANGELIMIT option .....	231
9.1.6 The CHECKPAGE option .....	232
9.1.7 The SCOPE parameter .....	233
9.1.8 The CLONE option .....	235
9.1.9 Full or incremental copies .....	235
9.1.10 SHRLEVEL REFERENCE or SHRLEVEL CHANGE .....	236



9.1.11 Tape or disk .....	236
9.2 COPYTOCOPY .....	237
9.2.1 COPYTOCOPY options .....	239
9.3 Concurrent COPY .....	240
9.3.1 Concurrent COPY SHRLEVEL REFERENCE using FILTERDDN .....	244
9.4 DSN1COPY .....	247
9.4.1 Copying tables from one DB2 subsystem to another .....	248
<b>Chapter 10. Recovering data .....</b>	<b>251</b>
10.1 Recovering from backup .....	252
10.2 System level point-in-time recovery .....	252
10.2.1 Using a system-level backup .....	253
10.3 Recovering to a point in time with consistency .....	253
10.4 RECOVER using the LISTDEF command .....	260
10.5 RECOVER of NOT LOGGED table spaces .....	262
10.6 Recovering a table space that contains LOB or XML data .....	262
10.7 RECOVER CLONE .....	263
10.8 Parallel RECOVER .....	263
10.8.1 Retaining tape mounts .....	265
10.8.2 Restrictions of the RECOVER utility with keyword PARALLEL .....	266
10.9 Recovering a data set or partition using DSNUM .....	267
10.9.1 Concurrency and compatibility .....	267
10.9.2 Adding partitions and recovery .....	268
10.9.3 Rotating partitions and recovery .....	268
10.9.4 Rebalancing partitions with ALTER, REORG, and recovery .....	268
10.10 Performance improvements for RECOVER with concurrent copies .....	268
10.11 LOGONLY .....	269
10.11.1 Copy taken outside DB2 .....	269
10.11.2 DB2 object names with REORG and FASTSWITCH .....	271
10.11.3 Recovering a DB2 object from CONCURRENT COPY .....	271
10.11.4 Recovering without an image copy .....	272
10.11.5 Recovering a single piece of a multi-linear page set .....	272
10.11.6 LOGONLY recovery improvements .....	272
10.12 LOGAPPLY considerations .....	273
10.13 Fast Log Apply .....	273
10.13.1 Fast Log Apply buffers .....	274
10.13.2 Sort the log records .....	275
10.13.3 When is Fast Log Apply bypassed .....	276
10.13.4 FLA performance .....	276
10.14 Avoiding specific image copy data sets .....	276
10.15 Restarting RECOVER .....	276
10.16 Displaying progress of RECOVER .....	276
10.17 Index recovery from COPY .....	277
10.18 PIT to a different DB2 function mode .....	280
10.19 General performance recommendations for RECOVER .....	280
10.20 MODIFY RECOVERY .....	281
10.20.1 RETAIN LAST(n) .....	284
10.20.2 RETAIN LOGLIMIT .....	284
10.21 REPORT RECOVERY .....	284
10.22 DB2 Recovery Expert .....	287
10.23 DB2 Change Accumulation Tool .....	288
10.24 QUIESCE .....	288
10.24.1 DPSI drain class and states considerations .....	290

10.24.2	Informational RI and QUIESCE. . . . .	290
10.24.3	QUIESCE NOT LOGGED considerations. . . . .	290
<b>Chapter 11.</b>	<b>Gathering statistics . . . . .</b>	<b>291</b>
11.1	Why collect statistics . . . . .	292
11.2	RUNSTATS. . . . .	292
11.2.1	DSTATS . . . . .	294
11.2.2	HISTORY . . . . .	296
11.2.3	Histograms . . . . .	299
11.3	When to run RUNSTATS . . . . .	300
11.4	RUNSTATS options . . . . .	303
11.4.1	REPORT . . . . .	303
11.4.2	LIST . . . . .	303
11.4.3	COLGROUP . . . . .	303
11.4.4	KEYCARD. . . . .	304
11.4.5	FREQVAL . . . . .	305
11.4.6	SAMPLE . . . . .	306
11.4.7	HISTOGRAM . . . . .	310
11.4.8	UPDATE . . . . .	310
11.4.9	HISTORY . . . . .	311
11.4.10	FORCEROLLUP . . . . .	311
11.4.11	SORTDEVT . . . . .	311
11.4.12	SORTNUM . . . . .	312
11.5	Inline statistics for discarded rows . . . . .	312
11.6	Fast cached SQL statement invalidation. . . . .	312
11.7	Modify statistics. . . . .	313
11.8	RUNSTATS and LOB table spaces . . . . .	316
11.8.1	RUNSTATS on the base table space . . . . .	316
11.8.2	RUNSTATS on the index of the auxiliary table . . . . .	316
11.8.3	RUNSTATS on the LOB table space . . . . .	317
11.8.4	RUNSTATS and XML objects. . . . .	319
<b>Chapter 12.</b>	<b>Verifying data consistency . . . . .</b>	<b>321</b>
12.1	CHECK SHRLEVEL CHANGE . . . . .	323
12.1.1	Online CHECK INDEX . . . . .	323
12.1.2	Online CHECK DATA . . . . .	326
12.1.3	Online CHECK LOB . . . . .	328
12.2	Sequence when executing the CHECK utilities . . . . .	330
12.3	CHECK DATA of NOT LOGGED table spaces. . . . .	330
12.4	CHECK of LOB and XML data . . . . .	331
12.4.1	CHECK DATA. . . . .	331
12.4.2	CHECK INDEX . . . . .	333
12.4.3	CHECK LOB . . . . .	334
12.5	CHECK of clone tables . . . . .	335
<b>Chapter 13.</b>	<b>BACKUP and RESTORE SYSTEM utilities . . . . .</b>	<b>339</b>
13.1	DFSMSshm and DB2 BACKUP and RESTORE SYSTEM. . . . .	340
13.1.1	FlashCopy. . . . .	340
13.1.2	TOKEN . . . . .	341
13.1.3	COPYPOOL . . . . .	341
13.1.4	DUMPCCLASS . . . . .	341
13.2	BACKUP SYSTEM . . . . .	342
13.3	BACKUP SYSTEM to DASD. . . . .	346
13.4	BACKUP SYSTEM to tape . . . . .	348

13.4.1 DUMP .....	348
13.4.2 DUMPONLY .....	348
13.5 BACKUP SYSTEM additional options. ....	349
13.5.1 DATA ONLY .....	349
13.5.2 FORCE .....	350
13.5.3 TOKEN .....	351
13.6 New DSNZPARMs .....	351
13.7 Incremental FlashCopy .....	351
13.7.1 Incremental FlashCopy with versions .....	352
13.8 Object-level recovery .....	355
13.9 Scenarios of SYSTEM BACKUP and object-level recovery .....	356
13.10 Considerations for using BACKUP and RESTORE SYSTEM. ....	368
13.11 Other available improvements .....	370
13.12 RESTORE SYSTEM. ....	370
13.13 Backing up the log Copy Pool prior to system-level recovery .....	371
13.14 Conditional restart after restoring the log Copy Pool .....	372
<b>Chapter 14. Operational considerations</b> .....	<b>375</b>
14.1 Utility execution .....	376
14.1.1 Preparing for utility execution .....	376
14.1.2 During utility execution .....	382
14.1.3 Compatibility of utilities in restrictive states. ....	385
14.1.4 Stopped utilities. ....	385
14.1.5 Restarting utilities .....	385
14.2 Parallelism. ....	385
14.2.1 Factors that determine the degree of parallelism .....	385
14.2.2 Determining if parallelism is constrained. ....	386
14.3 Concurrency .....	387
14.3.1 Running utilities concurrently .....	387
14.4 Number of concurrent utility jobs. ....	388
14.5 Data sharing considerations .....	388
14.5.1 Submitting utility jobs .....	388
14.5.2 Terminating and restarting utility jobs .....	388
14.5.3 Concurrency .....	389
14.5.4 Utility performance in a DSG. ....	389
14.5.5 Mixed releases in a DSG .....	389
14.6 Buffer pool considerations. ....	393
14.6.1 Page fix option .....	394
14.6.2 Buffer pool write operations .....	394
14.7 Specific utility considerations .....	394
14.7.1 SORT considerations .....	394
14.7.2 REORG considerations. ....	395
14.7.3 LOAD considerations .....	397
14.7.4 UNLOAD utility .....	397
14.7.5 COPY considerations .....	397
14.7.6 RECOVER utility .....	398
14.7.7 REPAIR utility .....	399
14.8 Miscellaneous considerations .....	400
14.8.1 Logging considerations .....	400
14.8.2 Invalidate cached dynamic statements .....	401
14.9 DSNZPARMs that affect utility execution .....	402
14.9.1 DSN6SPRM .....	402
14.9.2 DSN6SYSP .....	407

<b>Part 4. Appendixes</b>	411
<b>Appendix A. DB2 Tools</b>	413
A.1 DB2 Administration Tool	414
A.1.1 Generate LISTDEFS and TEMPLATES for utility generation	414
A.1.2 Generate ad hoc DB2 utilities	415
A.1.3 Generate offline DB2 utilities	418
A.1.4 Migrate DB2 catalog statistics	422
A.1.5 DB2 Performance Queries	423
A.2 DB2 Utility Enhancement Tool for z/OS	429
A.2.1 ISPF user interface	429
A.2.2 Batch processing	429
A.2.3 REORG utility options	429
A.2.4 Load utility options	432
A.2.5 CHECK DATA utility	435
A.3 DB2 Automation Tool	435
A.3.1 Object profile	436
A.3.2 Utility profile	439
A.3.3 Exception profile	443
A.3.4 Job profile	448
A.4 DB2 High Performance Unload for z/OS	455
A.5 DB2 Recovery Expert for z/OS	456
A.5.1 Recovery Expert features	456
A.5.2 BACKUP SYSTEM and RESTORE SYSTEM	457
A.6 DB2 Change Accumulation Tool	459
A.7 DB2 Storage Management Utility	459
A.8 Optim Query Tuner and Optim Query Workload Tuner	460
A.9 DB2 Buffer Pool Analyzer for z/OS	463
<b>Appendix B. Utilities related maintenance</b>	465
<b>Related publications</b>	469
IBM Redbooks	469
Other publications	469
Online resources	470
How to get Redbooks	470
Help from IBM	471
<b>Index</b>	473

# Figures

2-1 Partitioning index versus data-partitioned secondary index . . . . .	30
2-2 DPSI versus NPIs . . . . .	32
2-3 DB2 utilities and PBG concurrency . . . . .	34
2-4 Utilities and PBGs . . . . .	35
3-1 LISTDEF scenario . . . . .	51
3-2 LISTDEF expansion . . . . .	52
3-3 Example disposition with TEMPLATES . . . . .	57
3-4 Automatic data set sizing for the REORG utility . . . . .	60
3-5 TEMPLATE and LISTDEF combined (V6 shown for comparison) . . . . .	67
3-6 Utility/LISTDEF/TEMPLATE cross-reference . . . . .	68
3-7 OPTIONS functions . . . . .	72
3-8 Utility support for PREVIEW . . . . .	73
4-1 Copy a list of DB2 objects in parallel . . . . .	80
4-2 Recovering a list of DB2 objects in parallel . . . . .	81
4-3 LOAD syntax for activating parallel partition loading . . . . .	83
4-4 Partition parallel LOAD with PIB . . . . .	84
4-5 Rebuilding a partitioned index with PIB . . . . .	87
4-6 Rebuilding a non-partitioned index with PIB . . . . .	87
4-7 Rebuilding all indexes of a partitioned table space with PIB using SORTKEYS . . . . .	88
4-8 Rebuilding all indexes of a non-partitioned table space with PIB using SORTKEYS . . . . .	89
5-1 Non-clustering and clustering order . . . . .	98
5-2 Remove indirect row reference . . . . .	99
5-3 INDREF is correlated with cluster count . . . . .	101
5-4 INDREF is not correlated with cluster count . . . . .	102
5-5 OFFPOS and CLUSTERRATIO correlation . . . . .	103
5-6 Logical and physical view before reorganization . . . . .	108
5-7 Logical and physical view after reorganization . . . . .	109
5-8 LEAFDIST calculation . . . . .	110
5-9 Administrative Task Scheduler . . . . .	137
6-1 Comparing memory limit methods . . . . .	154
6-2 Parallelism without the SORTNUM elimination feature . . . . .	163
6-3 Parallelism with the SORTNUM elimination feature . . . . .	163
7-1 Unload from image copies . . . . .	187
7-2 LOB architecture . . . . .	190
7-3 File references for LOBs stored in a PDS or PDSE . . . . .	191
7-4 File references for LOBs stored on a HFS directory . . . . .	192
7-5 Unloading LOBs . . . . .	193
7-6 REORG SHRLEVEL CHANGE . . . . .	198
7-7 Interacting with BatchPipes . . . . .	203
8-1 REORG TABLESPACE PART n SHRLEVEL CHANGE phases in DB2 V8 . . . . .	215
8-2 REORG TABLESPACE PART n phases in DB2 9 . . . . .	216
8-3 LOB architecture . . . . .	217
8-4 Chunks of LOB pages . . . . .	218
9-1 Parallel COPY with three subtasks . . . . .	228
9-2 COPYTOCOPY . . . . .	237
10-1 Recovery to PIT with consistency highlights . . . . .	256
10-2 RECOVER with PARALLEL keyword . . . . .	265
10-3 DB2 installation screen DSNTIPL . . . . .	274

10-4	Best practices for MODIFY RECOVERY DATE .....	283
10-5	New MODIFY RECOVERY options .....	283
10-6	Best practices for MODIFY RECOVERY RETAIN .....	284
11-1	Statistics gathered by RUNSTATS .....	293
11-2	KEYCARD versus distribution statistics .....	295
11-3	Monitoring space growth with RUNSTATS HISTORY .....	298
11-4	Using PAGESAVE to determine dictionary effectiveness.....	299
11-5	RUNSTATS Histogram Statistics .....	300
11-6	MODIFY STATISTICS syntax.....	313
11-7	LOB catalog statistics .....	318
12-1	Online CHECK INDEX .....	324
12-2	Online CHECK DATA .....	327
12-3	Online CHECK LOB .....	329
13-1	Integration between database and storage.....	340
13-2	FlashCopy relationship .....	341
13-3	Data set layout considerations .....	343
13-4	Incremental FlashCopy with a single version .....	353
13-5	Incremental FlashCopy with two versions.....	354
13-6	Scenario 1.....	356
13-7	Scenario 2.....	358
13-8	Scenario 3.....	359
13-9	Scenario 4.....	360
13-10	Scenario 5.....	361
13-11	Scenario 6.....	363
13-12	Scenario 7.....	364
13-13	Scenario 8.....	364
13-14	Scenario 9.....	366
A-1	Sample of Statistics Advisor usage .....	460

# Tables

1-1 Major changes in the last DB2 versions .....	15
2-1 Number of partition and partitioned table space sizes .....	28
2-2 Table space type by SEGSIZE, MAXPARTITIONS, and Numparts values .....	32
3-1 Object results table by keyword .....	48
3-2 Pattern-matching character comparison table .....	50
3-3 TEMPLATE variables .....	58
3-4 Data disposition for dynamically allocated data sets .....	70
3-5 Data dispositions for dynamically allocated data sets on RESTART .....	71
5-1 CLUSTERRATIOF and DATAREPEATFACTOR impact .....	95
5-2 Limitations of the STATCLUS=STANDARD calculation .....	96
5-3 COPY CHANGLIMIT with a single value .....	104
5-4 COPY CHANGLIMIT with double values .....	104
5-5 Trigger limits and utilities .....	111
5-6 SYSINDEXES and SYSINDEXES_HIST .....	113
5-7 SYSINDEXPART and SYSINDEXPART_HIST .....	114
5-8 SYSTABLEPART and SYSTABLEPART_HIST .....	115
5-9 SYSTABLES and SYSTABLES_HIST .....	115
5-10 RTS indexes .....	118
5-11 How RTS tables are updated for DDL .....	119
5-12 How different utilities affect RTS tables .....	120
5-13 How SQL affects SYSTABLESPACESTATS .....	122
5-14 How SQL affects SYSINDEXSPACESTATS .....	122
5-15 New SYSTABLESPACESTATS columns .....	124
5-16 New SYSINDEXSPACESTATS columns .....	124
5-17 DB2 objects used by the Administrative Task Scheduler .....	138
6-1 Estimated main memory use per DFSORT task .....	154
10-1 Total number of processes used for RECOVER in parallel .....	264
10-2 DB2 catalog table entries .....	271
10-3 Fast Log Apply buffer sizes .....	274
10-4 NEW TTYPE values .....	285
11-1 Statistic tables updated by the HISTORY option .....	297
11-2 Allowable HISTORY/UPDATE combinations .....	298
11-3 Sampling performance .....	309
11-4 Deleting statistics gathered by HISTORY SPACE .....	314
11-5 Deleting statistics gathered by HISTORY ACCESSPATH .....	315
11-6 Deleting statistics gathered by HISTORY ALL .....	316
12-1 Automatically created objects .....	336
14-1 Storage allocated using REGION parameter: No IEFUSI exit installed .....	377
14-2 Utilities that can benefit from DFSMS striping .....	379
14-3 Mapping of partition identifiers and partition numbers .....	381
14-4 Data set name identifier and value .....	382
14-5 DB2 9 utility features and load module names .....	390
14-6 Writes per single I/O operation during utility execution .....	394
14-7 Scenario using DSN1COPY with different versions .....	400
14-8 DSNZPARMS for macro DSN6SPRM .....	402
14-9 DSNZPARMS for macro DSN6SYSP .....	407
A-1 Exception criteria .....	444
A-2 Different functions in OSC, OE, OQT, and OWQT .....	462

A-3	Optim products and links . . . . .	463
B-1	List of relevant maintenance for DB2 9 for z/OS . . . . .	465



# Examples

2-1 Create a table controlled partitioned table .....	31
2-2 Partition-by-growth CREATE TABLESPACE statement .....	33
2-3 Partition-by-growth CREATE Table statement .....	34
2-4 Create Partition By Range .....	38
3-1 Not using LISTDEF .....	49
3-2 Version 9 .....	49
3-3 Excluding objects .....	49
3-4 Sample LISTDEF: recovery job .....	51
3-5 Sample output .....	54
3-6 Template not invoking data set sizing .....	56
3-7 Template invoking data set sizing .....	56
3-8 Template using tape with STACK option .....	56
3-9 Using multiple TEMPLATES .....	56
3-10 Template with substring notation .....	59
3-11 Output job with substring .....	59
3-12 Use of keyword LIMIT .....	61
3-13 LIMIT keyword output .....	61
3-14 Unload LOB data to a PDS .....	63
3-15 Unload LOB data to a PDS .....	64
3-16 Template with UNIX support .....	65
3-17 Using LISTDEF and TEMPLATES together .....	69
3-18 Identify the character set used in the utility control statements .....	74
4-1 Initiating multiple inline COPY within LOAD utility .....	77
4-2 Initiating inline COPY when loading at a partition level .....	77
4-3 The monitoring DSNU397I message .....	91
5-1 LEAFDIST formula .....	110
5-2 Creating historical RTS tables .....	127
5-3 Cross loader for RTS history .....	127
5-4 Objects in SYSTABLESPACESTATS but not in TABLESPACESTATS _HIST .....	128
5-5 Objects that do not exist in SYSTABLESPACESTATS .....	128
5-6 Objects that do not exist in SYSINDEXSPACESTATS .....	129
5-7 SQL using REORGUNCLUSTERINS/TOTALROWS .....	130
5-8 DSNACCOX formula for recommending a REORG on a table space .....	136
6-1 Determining the sort parameters passed to DFSORT in the UTPRINnn message data set .....	146
6-2 Overriding the DB2 estimated FILSZ value .....	146
6-3 Verifying if the new FILSZ was used .....	147
6-4 List DFSORT installation defaults .....	148
6-5 DFSORT installation defaults .....	148
6-6 Determining the sort method used from the DFSORT message data sets .....	150
6-7 DFSORT ICEPRMxx parmlib member .....	150
6-8 Dynamically changing a DFSORT default option .....	151
6-9 List of the changed DFSORT parameters .....	151
6-10 Determining the storage used by an individual sort task before PK79136 .....	152
6-11 Determining the storage used by an individual sort task with PK79136/PK92248 ..	153
6-12 Sample JCL for invoking DSNUTILB .....	155
6-13 Get additional diagnostic messages out of DFSORT .....	156
6-14 DB2 utility abending because of DFSORT problems .....	156

6-15	Verifying the FILSZ and AVGREC value passed by DB2 with the actual number of rows to be sorted . . . . .	158
6-16	Determining the FILSZ parameter passed to DFSORT in the UTPRINnn message data set. . . . .	162
7-1	LOAD of a segmented table space with Parallel Index Build . . . . .	171
7-2	Job output of LOAD with Parallel Index Build . . . . .	172
7-3	Load of a partitioned table space with full parallelism. . . . .	172
7-4	Job output of LOAD with full parallelism . . . . .	173
7-5	UNLOAD statement to generate the input statements for LOAD . . . . .	174
7-6	Converting the row format. . . . .	177
7-7	Delimited format . . . . .	179
7-8	Use of COPYDICTIONARY to copy a compression dictionary . . . . .	181
7-9	Loading Unicode data . . . . .	182
7-10	Load LOG NO without inline copy. . . . .	184
7-11	LOAD LOG NO without the inline copy job output . . . . .	184
7-12	LOAD LOG NO NOCOPYPEND without inline copy job output . . . . .	184
7-13	LOAD of a NOT LOGGED segmented table space . . . . .	185
7-14	LOAD of a NOT LOGGED partitioned table space . . . . .	185
7-15	LOAD of a NOT LOGGED partitioned table space with NOCOPYPEND option . . .	186
7-16	Simple cross loader example . . . . .	195
7-17	Specifying SORTKEYS and SPACE estimations to the cross loader. . . . .	197
7-18	Using the PATH keyword to unload DB2 data into an HFS file. . . . .	200
7-19	Attributes of the HFS unload file . . . . .	201
7-20	Using the PATH keyword to load a HFS file to DB2 . . . . .	201
8-1	Reorganization of a segmented table space. . . . .	208
8-2	Job output of REORG with Parallel Index Build . . . . .	208
8-3	Reorganization of a partitioned table space with full parallelism. . . . .	209
8-4	Job output of REORG of a partitioned table space with full parallelism . . . . .	209
8-5	Creating a unique mapping table for &UTILID . . . . .	210
8-6	REORG SHRLEVEL REFERENCE of a partitioned table space with NOSYSREC . .	211
8-7	REORG SHRLEVEL REFERENCE of a partitioned table space without NOSYSREC	211
8-8	DDL of table DSN8910.PRODUCT containing a XML column . . . . .	220
8-9	Job output for REORG SHRLEVEL CHANGE of an XML table space. . . . .	220
9-1	Specifying lists in the COPY utility . . . . .	227
9-2	Parallel COPY using LISTDEF . . . . .	229
9-3	Output from Parallel COPY using LISTDEF . . . . .	229
9-4	-DISPLAY UTILITY for Parallel COPY . . . . .	230
9-5	Specifying CHANGELIMIT . . . . .	232
9-6	Report output from COPY utility . . . . .	232
9-7	Source and output for SCOPE PENDING . . . . .	234
9-8	Using CLONE option. . . . .	235
9-9	COPYTOCOPY and LISTDEFs . . . . .	239
9-10	COPYTOCOPY using FROMLASTCOPY . . . . .	240
9-11	Output job for Concurrent COPY with error . . . . .	241
9-12	Output from incremental copy after a Full Concurrent COPY. . . . .	242
9-13	Example of RECOVER using a Concurrent COPY. . . . .	243
9-14	Copy statement without FILTERDDN . . . . .	244
9-15	Copy statement using FILTERDDN . . . . .	244
9-16	Complete JCL using FILTERDDN . . . . .	245
9-17	Job output using FILTERDDN . . . . .	246
9-18	DSN1COPY output. . . . .	247
9-19	Example of problems with DSN1COPY . . . . .	248
10-1	RECOVER with consistency. . . . .	258

10-2 RECOVER TORBA . . . . .	259
10-3 RECOVER using a LISTDEF command with a wildcard. . . . .	260
10-4 LISTDEF OPTIONS PREVIEW job output . . . . .	261
10-5 Partial job output with seven objects in parallel . . . . .	264
10-6 RECOVER with object-list. . . . .	266
10-7 Recover four data sets using DSNUM . . . . .	267
10-8 DSNU116I message . . . . .	277
10-9 COPY table space and all indexes . . . . .	277
10-10 RECOVER of the table space and indexes in parallel . . . . .	278
10-11 RECOVER table space and indexes output . . . . .	278
10-12 REPORT RECOVERY sample job . . . . .	286
10-13 REPORT RECOVERY sample output . . . . .	286
10-14 QUIESCE with a list of table spaces. . . . .	288
10-15 QUIESCE using the LISTDEF command . . . . .	289
10-16 QUIESCE of table space and all indexes . . . . .	289
11-1 Check the status of the table space . . . . .	292
11-2 Statistics for SPACE and EXTENTS. . . . .	297
11-3 Data set listing . . . . .	297
11-4 A sample SQL to evaluate the frequency of changes. . . . .	302
11-5 RUNSTATS using KEYCARD. . . . .	304
11-6 Cardinality statistics across column sets with a three-column index. . . . .	305
11-7 Collect distribution statistics . . . . .	306
11-8 RUNSTATS SAMPLE 10 . . . . .	307
11-9 RUNSTATS SAMPLE 25 . . . . .	308
11-10 RUNSTATS SAMPLE 100 . . . . .	309
11-11 Command cached SQL statement invalidation. . . . .	312
11-12 MODIFY STATISTICS by date using SPACE. . . . .	314
11-13 Output job for table space. . . . .	314
11-14 RUNSTATS with LISTDEF for LOB table spaces. . . . .	318
11-15 RUNSTATS for LOB table spaces . . . . .	318
12-1 Invoking CHECK INDEX with SHRLEVEL CHANGE . . . . .	324
12-2 Output of CHECK INDEX with SHRLEVEL CHANGE . . . . .	325
12-3 When FlashCopy fails because of shadow data sets allocated on PPRC volumes . . . . .	326
12-4 Invoking CHECK DATA with SHRLEVEL CHANGE . . . . .	328
12-5 Invoking CHECK LOB with SHRLEVEL CHANGE . . . . .	329
12-6 DDL to create table with BLOB columns and CLONE . . . . .	335
12-7 VSAM objects created for table and clone table with LOB column. . . . .	336
12-8 Sample SQL to populate the base table . . . . .	336
12-9 Sample SQL to populate the clone table . . . . .	337
12-10 Checking the base and clone object with the DISPLAY DATABASE command. . . . .	337
13-1 DB2 DDF display . . . . .	343
13-2 List of Copy Pools name . . . . .	344
13-3 List of storage groups . . . . .	344
13-4 Volumes for storage group . . . . .	345
13-5 LIST COPYPOOL job . . . . .	345
13-6 Relationship between source and target volumes . . . . .	346
13-7 BACKUP SYSTEM JCL utility. . . . .	346
13-8 BACKUP SYSTEM output . . . . .	347
13-9 Output of BSDS . . . . .	347
13-10 DFSMSshm message on MVS log . . . . .	349
13-11 Output for DUMP parameter. . . . .	349
13-12 BACKUP FORCE . . . . .	350
13-13 TOKEN parameter . . . . .	351

13-14	Example of Incremental BACKUP SYSTEM. . . . .	352
13-15	Output for Incremental BACKUP SYSTEM . . . . .	352
13-16	Output from HSM Copy Pool . . . . .	353
13-17	Output for backups . . . . .	356
13-18	Output of SYSTEM BACKUP . . . . .	356
13-19	Incremental system backup output . . . . .	357
13-20	Output of RECOVER . . . . .	357
13-21	Output of RECOVER . . . . .	358
13-22	Output from BSDS . . . . .	358
13-23	Output for Recovery Logpoint. . . . .	359
13-24	DB2 output for recovery . . . . .	360
13-25	DFSMSshm messages when DASD was changed. . . . .	360
13-26	Recovering output for the new table space created . . . . .	361
13-27	BACKUP SYSTEM FULL output. . . . .	362
13-28	Backup full output and recovery for table space with referential integrity . . . . .	362
13-29	Last incremental FlashCopy output . . . . .	363
13-30	Recovery output . . . . .	363
13-31	Recovery index output . . . . .	364
13-32	DFSMSshm output . . . . .	365
13-33	DFSMSshm output . . . . .	365
13-34	DB2 recovery utility job . . . . .	366
13-35	Output for SMS command and fail recovery job . . . . .	366
13-36	Output for DUMPONLY. . . . .	367
13-37	HSM messages on the MVS log . . . . .	367
13-38	LIST COPYPOOL job . . . . .	367
13-39	Output LIST COPYPOOL . . . . .	368
13-40	FRBACKUP JCL . . . . .	371
13-41	DFSMSshm output . . . . .	371
13-42	DFSMSshm list report. . . . .	372
14-1	DEFINE CLUSTER using MODEL parameter . . . . .	380
14-2	Query for instance qualifier for a table space . . . . .	381
14-3	Query for instance qualifier for an index space. . . . .	381
14-4	Query against SYSTABLEPART . . . . .	382
14-5	Query against SYSINDEXPART. . . . .	382
14-6	DISPLAY UTILITY output . . . . .	383
14-7	SYSPRINT messages depicting Julian date and timestamp . . . . .	384
14-8	Modification to DSNUPROC for release coexistence . . . . .	391
14-9	Cross-copy JCL procedure . . . . .	393
14-10	DISPLAY CLAIMERS output . . . . .	396
14-11	LOAD COPYDICTIONARY. . . . .	397
A-1	Administration main menu . . . . .	414
A-2	Template example for UNLOAD data set. . . . .	415
A-3	DB2 Table Spaces . . . . .	416
A-4	Table Space Utilities screen. . . . .	417
A-5	Copy utility options . . . . .	418
A-6	SP command . . . . .	419
A-7	Table Space Parts screen . . . . .	419
A-8	Table Space Utilities screen: DSN1 option. . . . .	420
A-9	Offline Utilities Selection screen . . . . .	421
A-10	Migrate catalog statistics . . . . .	422
A-11	DB2 Performance Queries . . . . .	423
A-12	Performance query option 7 . . . . .	424
A-13	Performance query option 14: table space maintenance recommendations . . . . .	425

A-14	Option 14: Table Space Maintenance . . . . .	426
A-15	Performance query option 14X: parameters for RTS Index Space Maintenance. . .	427
A-16	Option 14X: Index Space maintenance . . . . .	428
A-17	Online REORG output using DB2 UET . . . . .	430
A-18	SYSPRINT for PRESORT option . . . . .	432
A-19	PRESORT and CONSTANT options . . . . .	433
A-20	VALUEIF option . . . . .	434
A-21	DB2 Automation Tool main menu. . . . .	435
A-22	Object profile display . . . . .	436
A-23	Adding objects to an Object Profile . . . . .	436
A-24	Partitioned table space handling . . . . .	437
A-25	Summary of objects in an Object Profile . . . . .	438
A-26	Explode current object list . . . . .	438
A-27	Utility profile options . . . . .	439
A-28	Image copy options . . . . .	440
A-29	Image copy options 2 . . . . .	441
A-30	Image copy options 3 . . . . .	442
A-31	Image copy options 4 . . . . .	443
A-32	Exception profile . . . . .	444
A-33	Job profile display . . . . .	448
A-34	Override setup options . . . . .	449
A-35	Template and listdef options. . . . .	449
A-36	Job breakdown options. . . . .	450
A-37	Sample job profile screen . . . . .	450
A-38	Build job screen . . . . .	451
A-39	The actual generated JCL . . . . .	451
A-40	Generation of JCL for utilities . . . . .	453



# Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

*IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.*

**The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:** INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

## COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.


# Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at <http://www.ibm.com/legal/copytrade.shtml>

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

BatchPipes®  
CICS®  
DataStage®  
DB2®  
DRDA®  
DS8000®  
FlashCopy®  
HyperSockets™

Hiperspace™  
IBM®  
IMS™  
InfoSphere™  
OMEGAMON®  
Optim™  
OS/390®  
Parallel Sysplex®

RACF®  
Redbooks®  
Redpapers™  
Redbooks (logo) ®  
RETAIN®  
System z®  
Tivoli®  
z/OS®

The following terms are trademarks of other companies:

Snapshot, and the NetApp logo are trademarks or registered trademarks of NetApp, Inc. in the U.S. and other countries.

Oracle, JD Edwards, PeopleSoft, Siebel, and TopLink are registered trademarks of Oracle Corporation and/or its affiliates.

ACS, and the Shadowman logo are trademarks or registered trademarks of Red Hat, Inc. in the U.S. and other countries.

SAP, and SAP logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries.

Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.



# Preface

IBM® continues to enhance the functionality, performance, availability, and ease of use of IBM DB2® utilities.

This IBM Redbooks® publication is the result of a project dedicated to the current DB2 Version 9 Utilities Suite product. It provides information about introducing the functions that help set up and invoke the utilities in operational scenarios, shows how to optimize concurrent execution of utilities and collect information for triggering utilities execution, and provides considerations about partitioning.

It also describes the new functions provided by several utilities for SHARE LEVEL CHANGE execution, which maximize availability and the exploitation of DFSMS constructs by the BACKUP and RESTORE SYSTEM utilities.

This book concentrates on the enhancements provided by DB2 UDB for z/OS Version 8 and DB2 for z/OS Version 9. It implicitly assumes a basic level of familiarity with the utilities provided by DB2 for z/OS and OS/390® Version 7.

## The team who wrote this book

This book was produced by a team of specialists from around the world working at the International Technical Support Organization, San Jose Center.

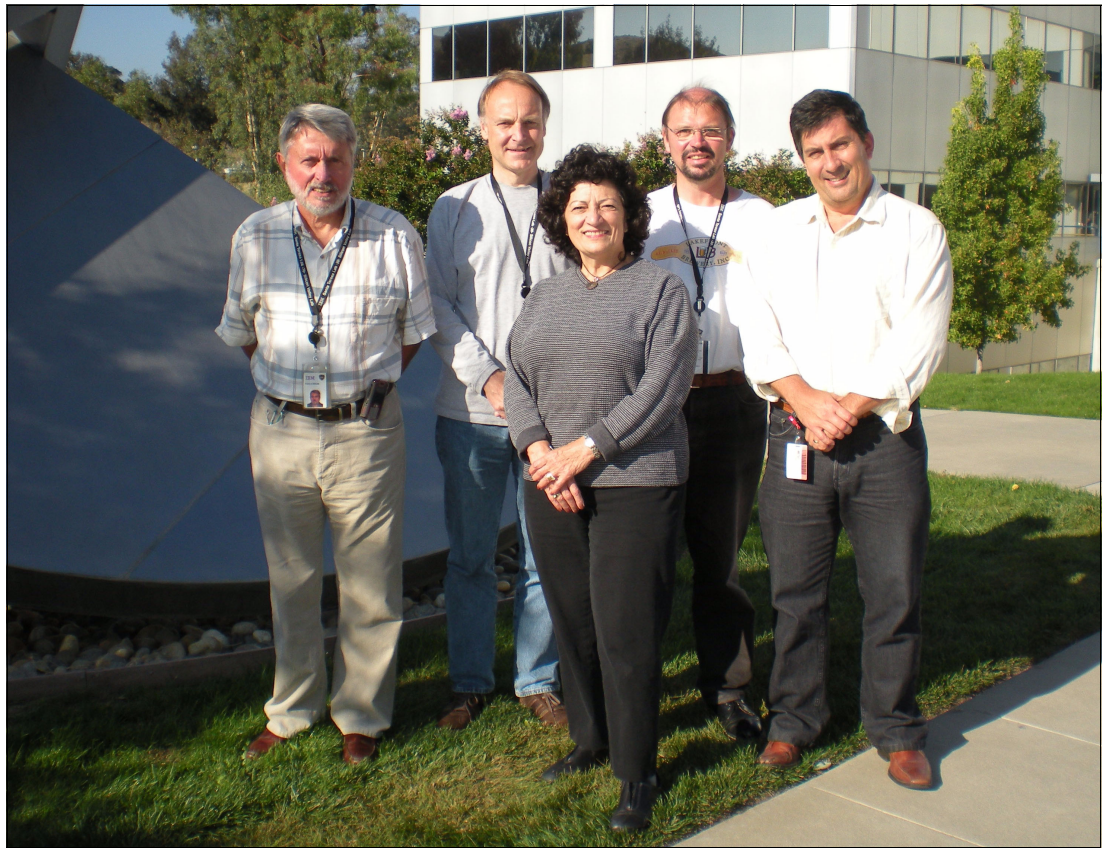
**Paolo Bruni** is a DB2 Information Management Project Leader at the International Technical Support Organization based in Silicon Valley Lab, San Jose, California. In this capacity, he has authored several IBM Redbooks publications on DB2 for z/OS and Data Management tools, and has conducted workshops and seminars worldwide. During Paolo's many years with IBM, in development and in the field, his work has been mostly related to database systems.

**Marcelo Antonelli** is a DB2 Specialist with IBM Brazil. He has 20 years of experience working with DB2 for z/OS. Marcelo holds a graduate degree in System Analysis from PUCP in Campinas, Sao Paulo. His areas of expertise include database design, system administration, and performance. Marcelo is currently supporting an outsourcing internal IBM account from EUA, as well as assisting IBM technical professionals working with DB2. Marcelo co-authored the books *DB2 for z/OS and OS/390 Version 7 Using the Utilities Suite*, SG24-6289, *IBM DB2 Performance Expert for z/OS Version 2*, SG24-68677, *Administration Solutions for DB2 UDB for z/OS*, SG24-6685, and *A Deep Blue View of DB2 Performance: IBM Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS*, SG24-7224.

**Davy Goethals** is a Belgian systems engineer. He works for ArcelorMittal, the biggest steel producing company in the world. His experience as a DB2 system administrator goes back to DB2 Version 1 in 1985. He participated in multiple DB2 ESP and QPP programs from DB2 V2.1 up to DB2 V7. Davy co-authored the book *LOBs with DB2 for z/OS: Stronger and Faster*, SG24-7270. Currently he is the DB2 and IMS™ team leader within the ArcelorMittal z/OS® department, located in Dunkerque, France, and is responsible for supporting more than 40 DB2 systems for steel plants all over Europe. He is also a regular presenter at international DB2 conferences, such as IDUG.

**Armin Kompalka** is an IBM Certified Senior IT specialist working for IBM Software Group Germany on DB2 Tools on IBM System z®. He has over 20 years of experience in DB2, IMS, and CICS® system programming and distributed database administration on various UNIX® platforms. He joined IBM in 2000 after working for 6 years with BMC as a technical DB2 tools specialist. Before that, he had worked as a DB/DC Systems programmer for Dun and Bradstreet. Armin co-authored the books *IMS DataPropagator Implementation Guide*, SG24-6838 and *SAP Casebook: DB2 Backup, Recovery and Cloning for SAP Environments*.

**Mary Petras** is a Consulting Product Design Professional at the IBM Silicon Valley Lab and works on the DB2 Tools Technical Specialist team supporting DB2 for z/OS Tools. Mary holds a graduate degree in Mathematics from Pratt Institute, School of Engineering and Science, in Brooklyn, New York, and did post-graduate work at the University of Connecticut in Storrs. Her areas of expertise include DB2 data sharing, DB2 utilities, and performance and tuning. Mary co-authored the books *DB2 UDB for OS/390 Version 6 Performance Topics*, SG24-5351, *IBM DB2 Performance Expert for z/OS Version 2*, SG24-6867, and *DB2 UDB for z/OS Version 8 Performance Topics*, SG24-6465. She has also presented at the DB2 Technical Conference and IDUG and local user groups.



*The authors in Janet Perna Plaza in SVL: From left to right: Paolo, Armin, Mary, Davy, and Marcelo*

Thanks to the following people for their contributions to this project:

Heidi Arnold  
David Betten  
Chuck Bonner  
Ben Budiman  
Marc Casad  
Karelle Cornwell  
Craig Friske  
Arsia Ghahari  
Oliver Harm  
Koshy John  
Laura Kunioka-Weis  
Xi Lia  
Christian Michel  
Roger Miller  
Ka-Chun Ng  
Haakon Roberts  
Vicky Vezinaw  
Frances Villafuerte  
Jie X Zhang  
Timm Zimmermann  
**IBM SVL**

Larry Jardine  
**Aetna USA**

Rick Butler  
**BMO Financial Group, Toronto**

Ernie Mancill  
Hennie Mynhardt  
**IBM USA**

Rich Conway  
Bob Haimowitz  
Emma Jacobs  
**International Technical Support Organization**

Thanks to the authors of the previous editions of this book:

The authors of the first edition, *DB2 for z/OS and OS/390 Version 7 Using the Utilities Suite*, SG24-6289, published in August 2001, were:

Paolo Bruni  
Marcelo Antonelli  
Tom Crocker  
Davy Goethals  
Rama Naidoo  
Bart Steegmans

## Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author - all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

[ibm.com/redbooks/residencies.html](http://ibm.com/redbooks/residencies.html)

## Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:

[ibm.com/redbooks](http://ibm.com/redbooks)

- Send your comments in an e-mail to:

[redbooks@us.ibm.com](mailto:redbooks@us.ibm.com)

- Mail your comments to:

IBM Corporation, International Technical Support Organization  
Dept. HYTD Mail Station P099  
2455 South Road  
Poughkeepsie, NY 12601-5400

# Summary of changes

This section describes the technical changes made in this edition of the book and in previous editions. This edition may also include minor corrections and editorial changes that are not identified.

Summary of Changes  
for SG24-6289-01  
for DB2 9 for z/OS: Using the Utilities Suite  
as created or updated on April 27, 2010.

## February 2010, Second Edition

The revisions of this Second Edition, first published on February 19, 2007, reflect the changes and additions described below.

## April 2010, First Update

This revision reflects the addition, deletion, or modification of new and changed information described below.

### Changed information

- ▶ Corrected values in Table 2-1 on page 28.
- ▶ Corrected the description of the algorithm at 2.4.3, “Rebalance partitions dynamically” on page 41. The purpose of REBALANCE has always been to balance the number of rows per partition. This was originally done on a page count based algorithm, then changed to a more precise row count based algorithm.
- ▶ Corrected some text at “Changing the partition boundaries with REORG REBALANCE” on page 43.
- ▶ Corrected the text of “Implications for PIT recovery” on page 43.
- ▶ Removed a sentence at “REORG non-consecutive partitions” on page 396.

### New information

- ▶ No new information was added.





# Part 1

## Introduction

In this short part, we introduce the DB2 utilities, summarize their evolution, and describe the contents of this book.

This part contains the following chapter:

- Chapter 1, “Introduction” on page 3







# Introduction

In this chapter, we provide an introduction to the contents of the book and a summary of the utilities available within the DB2 Utility Suite.

The chapter contains the following sections:

- ▶ Contents of this book
- ▶ Brief overview of all DB2 utilities
- ▶ Utilities at work
- ▶ Major changes since DB2 7

## 1.1 Contents of this book

In Part 1, “Introduction” on page 1 we briefly introduce all the DB2 utilities, summarize their evolution, and highlight the major changes that have occurred since DB2 V7.

In Part 2, “Planning for DB2 utilities” on page 25, we include overall considerations about functions related to planning for and executing the DB2 utilities. These considerations are distributed across five chapters and describe the following topics:

- ▶ Wildcarding and templates
- ▶ Performance via parallelism of executions
- ▶ Triggering and controlling executions
- ▶ Managing partitions
- ▶ Considerations on SORT usage

In Part 3, “Executing DB2 utilities” on page 165, we delve into the execution of the new DB2 utilities and the new functions by describing their options and providing examples of usage for:

- ▶ Loading data
- ▶ Reorganizing data
- ▶ Unloading data
- ▶ Recovering data
- ▶ Copying data
- ▶ Gathering statistics
- ▶ Using BACKUP and RESTORE
- ▶ Operational considerations

In Part 4, “Appendixes” on page 411, we include a brief overview of the IBM DB2 Tools that can assist you with utility generation or offer additional utility features, as well as a list of recent utility related maintenance.

## 1.2 Brief overview of all DB2 utilities

With the latest versions of the DB2 utilities, IBM committed to continuing investment in satisfying customer demand for decreased total cost of ownership, increased productivity, availability, nondisruptive operation, and better performance. Functional changes are provided in the following areas:

- ▶ Additional cost savings through increasing zIIP offload capabilities
- ▶ Functions to allow the utilities to run non-disruptively
  - Eliminate outages.
  - Improve performance.
  - Reduce CPU cost.
- ▶ Functions that add real value
- ▶ Reduction of complexity and improved automation

Additional data types and functions have been introduced by DB2 Version 9:

- ▶ BIGINT, BINARY data types
- ▶ XML
- ▶ File reference variables

The DB2 utilities functions have also been extended to support these new types and functions.

DB2 for z/OS utilities support the XML data type. The storage structure for XML data and indexes is similar to the storage structure for LOB data and indexes. As with LOB data, XML data is not stored in the base table space, but it is stored in an auxiliary table space that contains only XML data. The XML table spaces also have their own index spaces. Therefore, most of the implications of using utilities for manipulating, backing up, and restoring XML data and LOB data are similar.

There are two types of DB2 utilities: online utilities and stand-alone utilities. In the next sections, we briefly describe the functions of each utility in both groups.

## 1.2.1 Online utilities

DB2 online utilities run as standard batch jobs or stored procedures, and they require DB2 to be active. They do not run under the control of the terminal monitor program (TMP); they have their own attachment mechanisms. They invoke DB2 control facility services directly.

Invoke the online utilities using one of the following methods:

- ▶ Use the DB2 utilities window in DB2I. This method requires very little involvement with JCL. To use this method, you must have TSO and access to DB2.
- ▶ Use the DSNU CLIST command in TSO. This method requires minimal involvement with JCL. To use this method, you must have TSO and access to DB2.
- ▶ Use the supplied JCL procedure (DSNUPROC). This method involves working with or creating your own JCL.
- ▶ Use the EXEC statement to create the JCL data set yourself. This method involves working with or creating your own JCL.
- ▶ Use the DSNUTILS or DSNUTILU stored procedure. This method involves invoking online utilities from a DB2 application program.

### BACKUP SYSTEM utility

The online BACKUP SYSTEM utility invokes z/OS DFSMSHsm (Version 1 Release 7 or above) to copy the volumes on which the DB2 data and log information resides for either a DB2 subsystem or data sharing group. You can use BACKUP SYSTEM to copy all data for a single application (for example, when DB2 is the database server for a resource planning solution). All data sets that you want to copy must be SMS-managed data sets. You can subsequently run the RESTORE SYSTEM utility to recover the entire system or the RECOVER utility to recover a single table space. In a data sharing environment, if any failed or abnormally quiesced members exist, the BACKUP SYSTEM request fails.

The BACKUP SYSTEM utility uses Copy Pools. A Copy Pool is a defined set of storage groups that contain data that DFSMSHsm can back up and recover collectively. For more information about Copy Pools, see *z/OS V1R10.0 DFSMS Storage Administration Reference for DFSMSdfp, DFSMSdss, DFSMSHsm, SC26-7402*.

Each DB2 subsystem can have up to two Copy Pools, one for databases and one for logs. BACKUP SYSTEM copies the volumes that are associated with these Copy Pools at the time of the copy. With the BACKUP SYSTEM utility, you can manage the dumping of system-level backups (copy of the database, the log Copy Pools, or both) to tape. To use this functionality, you need to have z/OS DFSMSHsm V1R8 or above. To use the DISPLAY UTILITY command for BACKUP SYSTEM on a data sharing group, you must issue the command from the member on which the BACKUP SYSTEM utility was invoked. Otherwise, the current utility information is not displayed.

## CATENFM

The CATENFM utility enables a DB2 subsystem to enter DB2 Version 9.1 enable-new-function mode and Version 9.1 new-function mode. It also enables a DB2 subsystem to return to enabling-new-function mode from new-function mode. All new Version 9.1 functions are unavailable when the subsystem is in conversion mode or enabling-new-function mode.

## CATMAINT

The CATMAINT online utility updates the catalog; it should be run during migration to a new release of DB2, or when instructed to do so by IBM Support.

## CHECK DATA

The CHECK DATA utility checks table spaces for violations of referential and table check constraints, and reports information about violations that it detects. CHECK DATA checks for consistency between a base table space and the corresponding LOB or XML table spaces. CHECK DATA does not check LOB table spaces. The utility does not check informational referential constraints. Run CHECK DATA after a conditional restart or a point-in-time recovery on all table spaces where parent and dependent tables might not be synchronized or where base tables and auxiliary tables might not be synchronized.

**Consideration:** Do not run CHECK DATA on encrypted data. Because CHECK DATA does not decrypt the data, the utility might produce unpredictable results

## CHECK INDEX

The CHECK INDEX online utility tests whether indexes are consistent with the data that they index, and issues warning messages when it finds an inconsistency. Run the CHECK INDEX utility after a conditional restart or a point-in-time recovery on all table spaces whose indexes might not be consistent with the data. Also, run CHECK INDEX before running CHECK DATA, especially if you specify DELETE YES. Running CHECK INDEX before CHECK DATA ensures that the indexes that CHECK DATA uses are valid. When checking an auxiliary table index, CHECK INDEX verifies that each LOB is represented by an index entry, and that an index entry exists for every LOB.

**Important:** Inaccurate statistics for tables, table spaces, or indexes can result in a sort failure during CHECK INDEX.

## CHECK LOB

The CHECK LOB online utility can be run against a LOB table space to identify any structural defects in the LOB table space and any invalid LOB values. You run the CHECK LOB online utility against a LOB table space that is marked CHECK pending (CHKP) to identify structural defects. If none are found, the CHECK LOB utility turns the CHKP status off.

You can run the CHECK LOB online utility on a LOB table space to identify any structural defects in the LOB table space and any invalid LOB values. The CHECK LOB utility is useful in a variety of circumstances:

- ▶ Run the utility on a LOB table space that is in CHECK-pending (CHKP) status to identify structural defects. If none are found, the CHECK LOB utility turns the CHKP status off.
- ▶ Run the utility on a LOB table space that is in auxiliary-warning (AUXW) status to identify invalid LOBs. If none exist, the CHECK LOB utility turns AUXW status off.
- ▶ Run the utility after a conditional restart or a point-in-time recovery on all table spaces where LOB table spaces might not be synchronized.

- ▶ Run the utility before you run the CHECK DATA utility on a table space that contains at least one LOB column.

## **COPY**

The COPY online utility creates up to four image copies of any of the following objects: table space, table space partition, data set of a linear table space, index space, or index space partition. The two types of image copies are:

- ▶ A full image copy, which is a copy of all pages in a table space, partition, data set, or index space.
- ▶ An incremental image copy, which is a copy of only those data pages that have been modified since the last use of the COPY utility.

The RECOVER utility uses these copies when recovering a table space or index space to the most recent time or to a previous time. Copies can also be used by the MERGECOPY, RECOVER, COPYTOCOPY, and UNLOAD utilities. You can copy a list of objects in parallel to improve performance. Specifying a list of objects along with the SHRLEVEL REFERENCE option creates a single recovery point for that list of objects. Specifying the PARALLEL keyword allows you to copy a list of objects in parallel, rather than serially.

## **COPYTOCOPY**

The COPYTOCOPY online utility makes image copies from an image copy that was taken by the COPY utility. This includes Inline Copies made by the REORG or LOAD utilities. Starting with either the local primary or recovery site primary copy, COPYTOCOPY can make up to three copies of one or more of the local primary, local backup, recovery site primary, and recovery site backup copies.

The copies are used by the RECOVER utility when recovering a table space or index space to the most recent time or to a previous time. These copies can also be used by MERGECOPY, UNLOAD, and possibly a subsequent COPYTOCOPY execution.

The COPYTOCOPY utility makes image copies from an image copy that was taken by the COPY utility, including inline copies that the REORG or LOAD utilities make. Starting with either the local primary or recovery-site primary copy, COPYTOCOPY can make up to three copies of one or more of the following types of copies:

- ▶ Local primary
- ▶ Local backup
- ▶ Recovery site primary
- ▶ Recovery site backup

You cannot run COPYTOCOPY on concurrent copies. The RECOVER utility uses the copies when recovering a table space or index space to the most recent time or to a previous time. These copies can also be used by MERGECOPY, UNLOAD, and possibly a subsequent COPYTOCOPY execution.

## **DIAGNOSE**

The DIAGNOSE online utility generates information that is useful in diagnosing problems. It is intended to be used only under the direction of your IBM Support Center.

## **EXEC SQL**

The EXEC SQL utility control statement declares cursors or executes dynamic SQL statements. You can use this utility as part of the DB2 cross-loader function of the LOAD utility.

The cross-loader function enables you to use a single LOAD job to transfer data from one location to another location or from one table to another table at the same location. You can use either a local server or any DRDA®-compliant remote server as a data input source for populating your tables. Your input can even come from other sources besides DB2 for z/OS; you can use the IBM Information Integrator Federation feature for access to data from sources as diverse as Oracle and Sybase, as well as the entire DB2 family of database servers.

## LISTDEF

The LISTDEF utility enables you to group database objects into reusable lists. You can then specify these lists in other utility control statements to indicate that the utility is to process all of the items in the list.

You can use LISTDEF to standardize object lists and the utility control statements that refer to them. This standardization reduces the need to customize or alter utility job streams.

If you do not use lists and you want to run a utility on multiple objects, you must run the utility multiple times or specify an itemized list of objects in the utility control statement.

## LOAD

Use LOAD to load one or more tables of a table space. The LOAD utility loads records into the tables and builds or extends any indexes that are defined on them. If the table space already contains data, you can choose whether you want to add the new data to the existing data or replace the existing data. The loaded data is processed by any edit or validation routine that is associated with the table, and any field procedure that is associated with any column of the table. The LOAD utility ignores and does not enforce informational referential constraints.

## MERGECOPY

The MERGECOPY online utility merges image copies that the COPY utility produces, image copies that the COPYTOCOPY utility produces, or inline copies that the LOAD or REORG utilities produce. The utility can merge several incremental copies of a table space to make one incremental copy. It can also merge incremental copies with a full image copy to make a new full image copy. You cannot run MERGECOPY on concurrent copies. MERGECOPY operates on the image copy data sets of a table space, and not on the table space itself.

## MODIFY RECOVERY

The MODIFY utility with the RECOVERY option deletes records from the SYSIBM.SYSCOPY catalog table, related log records from the SYSIBM.SYSLGRNX directory table, and entries from the DBD, and recycles the version numbers<sup>1</sup> for reuse. You can remove records that were written before a specific date, you can remove records of a specific age, or you can ensure that a specified number of records is retained.

---

<sup>1</sup> When ALTERing objects, DB2 stores the range of used version numbers in the OLDEST\_VERSION and CURRENT\_VERSION columns of one or more of the following catalog tables: SYSIBM.SYSTABLESPACE, SYSIBM.SYSTABLEPART, SYSIBM.SYSINDEXES, or SYSIBM.SYSINDEXPART. For objects with the COPY YES attribute, MODIFY RECOVERY updates the OLDEST\_VERSION column of the appropriate catalog tables with the version number of the oldest version not yet applied to the entire object. Recycling of version numbers is required when all of the version numbers are being used.

You can delete records for an entire table space, partition, or data set. You should run MODIFY regularly to remove outdated information from SYSIBM.SYSCOPY and SYSIBM.SYSLGRNX. These tables, and particularly SYSIBM.SYSLGRNX, can become very large and take up a considerable amount of space. By deleting outdated information from these tables, you can help improve performance for processes that access data from these tables.

The MODIFY RECOVERY utility automatically removes the SYSCOPY and SYSLGRNX recovery records that meet the specified criteria for all indexes on the table space that were defined with the COPY YES attribute.

## **MODIFY STATISTICS**

The online MODIFY STATISTICS utility deletes unwanted statistics history records from the corresponding catalog tables. You can remove statistics history records that were written before a specific date, or you can remove records of a specific age. You can delete records for an entire table space, index space, or index. Run MODIFY STATISTICS regularly to clear outdated information from the statistics history catalog tables. By deleting outdated information from those tables, you can improve performance for processes that access data from those tables.

## **OPTIONS**

The online OPTIONS utility control statement specifies processing options that are applicable across many utility executions in a job step.

By specifying various options, you can:

- ▶ Preview utility control statements
- ▶ Preview LISTDEF or TEMPLATE definitions
- ▶ Override library names for LISTDEF lists or TEMPLATE definitions
- ▶ Specify how to handle errors during list processing
- ▶ Alter the return code for warning messages
- ▶ Restore all default options

You can repeat an OPTIONS control statement within the SYSIN DD statement. If you repeat the control statement, it entirely replaces any prior OPTIONS control statement.

## **QUIESCE**

The online QUIESCE utility establishes a quiesce point for a table space, partition, table space set, or list of table spaces and table space sets. A quiesce point is the current log RBA or log record sequence number (LRSN). QUIESCE then records the quiesce point in the SYSIBM.SYSCOPY catalog table. When using RECOVER to a prior point-in-time with consistency, a quiesce point is no longer essential when planning for point-in-time recoveries, because the objects will be recovered with transactional consistency (the objects will only contain data that has been committed). However, recovering objects to a quiesce point will be faster because no work has to be backed out. Also, you might still want to establish quiesce points for related sets of objects if there is a need to plan for a point-in-time recovery for the entire set. You can recover a table space to its quiesce point by using the RECOVER TABLESPACE utility. With the WRITE(YES) option, QUIESCE writes changed pages for the table spaces and their indexes from the DB2 buffer pool to disk.

## **REBUILD INDEX**

The REBUILD INDEX utility reconstructs indexes or index partitions from the table that they reference.

## RECOVER

The online RECOVER utility recovers a single object or a list of objects to the current state or to a previous point in time by restoring a copy and then applying log records. The largest unit of data recovery is the table space or index space; the smallest is the page. The RECOVER utility recovers an entire table space, index space, a partition or data set, pages within an error range, or a single page. You can recover data using image copies or a system-level backup as a recovery base. Point in time recovery with consistency automatically detects the uncommitted transactions running at the recover point in time and will roll back their changes on the recovered objects. So, after recovery, the objects will be left in a transactionally consistent state.

## REORG INDEX

The online REORG INDEX utility reorganizes an index space to improve access performance and reclaim fragmented space. You can specify the degree of access to your data during reorganization, and you can collect inline statistics by using the STATISTICS keyword. You can determine when to run REORG INDEX by using the LEAFDISTLIMIT catalog query option. If you specify the REPORTONLY option, REORG INDEX produces a report that indicates whether a REORG is recommended; in this case, a REORG is not performed. These options are not available for indexes on the directory.

## REORG TABLESPACE

The online REORG TABLESPACE utility reorganizes a table space to improve access performance and to reclaim fragmented space. In addition, the utility materializes pending schema changes and can optionally build compression dictionaries.

REORG can reorganize a single partition or range of partitions of a partitioned table space. You can specify the degree of access to your data during reorganization, and you can collect inline statistics by using the STATISTICS keyword. If you specify REORG TABLESPACE UNLOAD EXTERNAL, the data is unloaded in a format that is acceptable to the LOAD utility of any DB2 subsystem.

You can also delete rows during the REORG job by specifying the DISCARD option.

The formulas in the DSNACCOX stored procedure help you determine when to run a REORG (or RUNSTATS or COPY) on specific objects.

Run the REORG TABLESPACE utility on a LOB table space to help increase the effectiveness of prefetch. If you specify SHRLEVEL REFERENCE, a REORG of a LOB table space will make LOB pages continuous and reclaim physical space if applicable.

Do not execute REORG on an object if another DB2 member holds retained locks on the object or has long-running, non-committed applications that use the object.

You can use the DISPLAY GROUP command to determine whether a member's status is failed. You can use the DISPLAY DATABASE command with the LOCKS option to determine if locks are held.

## REPAIR

The online REPAIR utility repairs data. The data can be your own data or data that you would not normally access, such as space map pages and index entries. You use REPAIR to replace invalid data with valid data. One typical usage is for fixing VERSIONS after having moved data objects across DB2 subsystems with DSN1COPY.

Be careful when using REPAIR. Improper use can damage the data even further.



## REPORT

The REPORT utility provides information about table spaces, tables, and indexes. Use REPORT TABLESPACESET to find the names of all the table spaces and tables in a referential structure, including LOB table spaces. The REPORT utility also provides information about the LOB table spaces that are associated with a base table space. Use REPORT RECOVERY to find information that is necessary for recovering a table space, index, or a table space and all of its indexes.

## RESTORE SYSTEM

The RESTORE SYSTEM utility invokes z/OS DFSMSHsm (Version 1 Release 7 or above) to recover a DB2 subsystem or a data sharing group to a previous point in time. To perform the recovery, the utility uses data that is copied by the BACKUP SYSTEM utility. All data sets that you want to recover must be SMS-managed data sets. The RESTORE SYSTEM utility can be run from any member in a data sharing group, even one that is normally quiesced when any backups are taken. Any member in the data sharing group that is active at or beyond the log truncation point must be restarted, and its logs are truncated to the SYSPITR LRSN point. You can specify the SYSPITR LRSN point in the CRESTART control statement of the DSNJU003 (Change Log Inventory) utility. Any data sharing group member that is normally quiesced at the time the backups are taken and is not active at or beyond the log truncation point does not need to be restarted. To be able to use system-level backups that have been dumped to tape, the level of DFSMSHsm must be V1R8 or higher. By default, RESTORE SYSTEM recovers the data from the database Copy Pool during the RESTORE phase and then applies logs to the point in time at which the existing logs were truncated during the LOGAPPLY phase. The RESTORE utility does not restore logs from the log copy pool; this can be done outside of DB2 using DFSMSHsm.

## RUNSTATS

The RUNSTATS utility gathers summary information about the characteristics of data in table spaces, indexes, and partitions. DB2 records these statistics in the DB2 catalog and uses them to select access paths to data during the bind process. You can use these statistics to evaluate the database design and determine when table spaces or indexes must be reorganized. To obtain the updated statistics, you can query the catalog tables. The two formats for the RUNSTATS utility are RUNSTATS TABLESPACE and RUNSTATS INDEX. RUNSTATS TABLESPACE gathers statistics on a table space and, optionally, on tables, indexes, or columns; RUNSTATS INDEX gathers statistics only on indexes. RUNSTATS does not collect statistics for clone tables or index spaces.

## STOSPACE

The STOSPACE utility updates DB2 catalog columns that indicate how much space is allocated for storage groups and related table spaces and indexes. For user-defined spaces, STOSPACE does not record any statistics.

## TEMPLATE

The TEMPLATE utility control statement lets you allocate data sets, without using JCL DD statements, during the processing of a LISTDEF list. In its simplest form, the TEMPLATE control statement defines the data set naming convention. You can also write TEMPLATE control statements so that they contain allocation parameters that define data set size, location, and attributes. Templates enable you to standardize data set names across the DB2 subsystem and to easily identify the data set type when you use variables in the data set name.

## UNLOAD

The online UNLOAD utility unloads data from one or more source objects to one or more BSAM sequential data sets in external formats. The source can be DB2 table spaces or DB2 image copy data sets. The source cannot be a concurrent copy. UNLOAD must be run on the system where the definitions of the table space and the table exists. UNLOAD is an enhancement of the REORG UNLOAD EXTERNAL function. With UNLOAD, you can unload rows from an entire table space or select specific partitions or tables to unload. You can also select columns by using the field specification list. If a table space is partitioned, you can unload all of the selected partitions into a single data set, or you can unload each partition in parallel into physically distinct data sets. The output records that the UNLOAD utility writes are compatible as input to the LOAD utility; as a result, you can reload the original table or different tables.

### 1.2.2 Stand-alone utilities

The stand-alone utilities execute as batch jobs independent of DB2. They can be executed only by using JCL. You can create utility control statements and EXEC PARM parameters for invoking the stand-alone utilities.

#### DSNJCNVB

The DSNJCNVB conversion utility converts the bootstrap data set (BSDS) so that it can support up to 10,000 archive log volumes and 93 active log data sets per log copy.

Running DSNJCNVB is mandatory when migrating from Version 8. DB2 Version 9.1 for z/OS will not start if the BSDS is in the old format. DSNJCNVB can be run any time after a Version 8 system has migrated to new function mode. Prior to converting the BSDS to the new format, it can manage only 1,000 archive log volumes and 31 active log data sets per log copy. Converting the BSDS is optional up until the migration to DB2 Version 9.1 for z/OS. DB2 must be stopped when running DSNJCNVB.

#### DSNJLOGF

When writing to an active log data set for the first time, DB2 must preformat a VSAM control area before writing the log records. The DSNJLOGF utility avoids this delay by preformatting the active log data sets before bringing them online to DB2.

#### DSNJU003

The DSNJU003 stand-alone utility changes the bootstrap data sets (BSDSs). You can use the utility to:

- ▶ Add or delete active or archive log data sets
- ▶ Add or delete checkpoint records
- ▶ Create a conditional restart control record to control the next start of the DB2 subsystem
- ▶ Change the VSAM catalog name entry in the BSDS
- ▶ Modify the communication record in the BSDS
- ▶ Modify the value for the highest-written log RBA value (relative byte address within the log) or the highest-offloaded RBA value

## **DSNJU004**

The DSNJU004 (print log map) utility lists a variety of information. The print log map (DSNJU004) utility lists the following information:

- ▶ Log data set name, log RBA association, and log LRSN for both copy 1 and copy 2 of all active and archive log data sets
- ▶ Active log data sets that are available for new log data
- ▶ Status of all conditional restart control records in the bootstrap data set
- ▶ Contents of the queue of checkpoint records in the bootstrap data set
- ▶ The communication record of the BSDS, if one exists
- ▶ Contents of the quiesce history record
- ▶ System and utility timestamps
- ▶ Contents of the checkpoint queue
- ▶ Archive log command history
- ▶ BACKUP SYSTEM utility history
- ▶ System CCSID information
- ▶ System-level backup information

In a data sharing environment, the DSNJU004 utility can list information from any or all BSDSs of a data sharing group.

## **DSN1CHKR**

The DSN1CHKR utility verifies the integrity of the DB2 directory and catalog table spaces. DSN1CHKR scans the specified table space for broken links, broken hash chains, and records that are not part of any link or chain. Use DSN1CHKR on a regular basis to promptly detect any damage to the catalog and directory.

DSN1CHKR is a diagnosis tool; it executes outside the control of DB2. You should have detailed knowledge of DB2 data structures to make proper use of this utility.

## **DSN1COMP**

The DSN1COMP utility estimates space savings to be achieved by DB2 data compression in table spaces and indexes. Because XML table spaces are range-partitioned universal table spaces or partition-by-growth universal table spaces, DSN1COMP fully supports them. This utility can be run on the following types of data sets containing uncompressed data:

- ▶ DB2 full image copy data sets
- ▶ VSAM data sets that contain DB2 table spaces or indexes
- ▶ Sequential data sets that contain DB2 table spaces (for example, DSN1COPY output) or indexes

DSN1COMP does not estimate savings for data sets that contain LOB table spaces.

## **DSN1COPY**

With the DSN1COPY stand-alone utility, you can copy:

- ▶ DB2 VSAM data sets to sequential data sets
- ▶ DSN1COPY sequential data sets to DB2 VSAM data sets
- ▶ DB2 image copy data sets to DB2 VSAM data sets
- ▶ DB2 VSAM data sets to other DB2 VSAM data sets
- ▶ DSN1COPY sequential data sets to other sequential data sets

A DB2 VSAM data set is a single piece of a non-partitioned table space or index, or a single partition of a partitioned table space or index. The input must be a single z/OS sequential or VSAM data set. Concatenation of input data sets is not supported. Using DSN1COPY, you can also print hexadecimal dumps of DB2 data sets and databases, check the validity of data or index pages (including dictionary pages for compressed data), translate database object identifiers (OBIDs) to enable moving data sets between different systems, and reset to 0 the log RBA that is recorded in each index page or data page. However, FlashCopies are supported by DSN1COPY.

You cannot run DSN1COPY on concurrent copies.

DSN1COPY can operate on both base and clone objects.

You can use the DSN1COPY utility on LOB table spaces by specifying the LOB keyword and omitting the SEGMENT and INLCOPY keywords.

## DSN1LOGP

The DSN1LOGP utility formats the contents of the recovery log for display. The two recovery log report formats are:

- ▶ A *detail report* of individual log records. This information helps IBM Support Center personnel analyze the log in detail. (This book does not include a full description of the detail report.)
- ▶ A *summary report* helps you:
  - Perform a conditional restart
  - Resolve indoubt threads with a remote site
  - Detect problems with data propagation

You can specify the range of the logs to process and select criteria within the range to limit the records in the detail report. For example, you can specify:

- ▶ One or more units of recovery identified by URID
- ▶ A single database

By specifying a URID and a database, you can display recovery log records that correspond to the use of one database by a single unit of recovery.

DSN1LOGP can print the log records for both base and clone table objects.

DSN1LOGP cannot read logs that have been compressed by DFSMS. (This compression requires extended format data sets.)

## DSN1PRNT

With the DSN1PRNT stand-alone utility, you can print:

- ▶ DB2 VSAM data sets that contain table spaces or index spaces (including dictionary pages for compressed data)
- ▶ Image copy data sets
- ▶ Sequential data sets that contain DB2 table spaces or index spaces

A DB2 VSAM data set is a single piece of a non partitioned table space or index, or a single partition of a partitioned table space or index. The input must be a single z/OS sequential or VSAM data set. Concatenation of input data sets is not supported.

Using DSN1PRNT, you can print hexadecimal dumps of DB2 data sets and databases. If you specify the FORMAT option, DSN1PRNT formats the data and indexes for any page that does not contain an error that would prevent formatting. If DSN1PRNT detects such an error, it prints an error message just before the page and dumps the page without formatting. Formatting resumes with the next page. DSN1PRNT supports compressed data. Compressed records are printed in compressed format. DSN1PRNT is especially useful when you want to identify the contents of a table space or index. You can run DSN1PRNT on image copy data sets as well as table spaces and indexes. DSN1PRNT accepts an index image copy as input when you specify the FULLCOPY option.

DSN1PRNT is compatible with LOB table spaces when you specify the LOB keyword and omit the INLCOPY keyword.

You cannot run DSN1PRNT on concurrent copies. DSN1PRNT does not decrypt any encrypted data; the utility displays the data as is.

### DSN1SDMP

Under the direction of the IBM Support Center, you can use the IFC Selective Dump (DSN1SDMP) utility to:

- ▶ Force dumps when selected DB2 trace events occur
- ▶ Write DB2 trace records to a user-defined MVS data set

## 1.3 Utilities at work

IBM has always enhanced the functionality, performance, resource cost, availability, ease of use and total cost of ownership of the initial set of utilities. Starting with DB2 for MVS/ESA Version 4, the progression of changes has increased. Table 1-1 summarizes the changes that occurred since V7 of DB2. Details about the functions and related performance of DB2 versions are reported in *DB2 for MVS/ESA Version 4 Non-Data-Sharing Performance Topics*, SG24-45622, *DB2 for OS/390 Version 5 Performance Topics*, SG24-2213, *DB2 UDB for OS/390 Version 6 Performance Topics*, SG24-5351, *DB2 for z/OS and OS/390 Version 7 Performance Topics*, SG24-6129, *DB2 UDB for z/OS Version 8 Performance Topics*, SG24-6465, and *DB2 9 for z/OS Performance Topics*, SG24-7473.

Table 1-1 Major changes in the last DB2 versions

Utility	DB2 V7	DB2 V8	DB2 9
All DB2 utilities	New packaging and dynamic utility jobs	<ul style="list-style-type: none"> <li>▶ The restartable online utility can be restarted with or without the RESTART keyword.</li> <li>▶ DFSORT integration.</li> <li>▶ SORTNUM elimination.</li> <li>▶ improved sorting estimates with Real Time Statistics.</li> <li>▶ DPSI Enhancements.</li> <li>▶ zIIP enablement for utility index processing.</li> </ul>	<ul style="list-style-type: none"> <li>▶ The utility messages have timestamp and Julian date.</li> <li>▶ Support for Universal, XML and Not logged Table Spaces.</li> <li>▶ Support for Clone Tables.</li> <li>▶ Index on Expression.</li> <li>▶ Support new data types.</li> <li>▶ Improved sorting estimates with Real Time Statistics.</li> <li>▶ New DSNACCOX stored procedure for utility recommendations.</li> </ul>

Utility	DB2 V7	DB2 V8	DB2 9
BACKUP SYSTEM	N/A	New utility BACKUP SYSTEM.	<ul style="list-style-type: none"> <li>▶ Tape control added.</li> <li>▶ Support of incremental FlashCopy®.</li> <li>▶ A feature to specify different storage classes on other utilities helps BACKUP SYSTEM.</li> </ul>
CHECK	N/A	INDEX SHRLEVEL CHANGE.	<ul style="list-style-type: none"> <li>▶ CHECK DATA and LOB SHRLEVEL CHANGE generate REPAIR statements.</li> <li>▶ Parallel index processing.</li> <li>▶ Storage class for shadow data sets.</li> <li>▶ Better information for (Q) replication is provided by the InfoSphere™ Replication Server product at utility termination.</li> <li>▶ SHRELEVEL REFERENCE parallel support.</li> </ul>
COPY	N/A	<ul style="list-style-type: none"> <li>▶ Online Concurrent Copy 32 KB pages support.</li> <li>▶ Performance PTFs for COPY of small DS to tape.</li> <li>▶ Performance PTFS.</li> </ul>	<ul style="list-style-type: none"> <li>▶ implicit CHECKPAGE.</li> <li>▶ SCOPE PENDING support.</li> <li>▶ Large block interface for tapes.</li> <li>▶ Template switching.</li> <li>▶ Large format dataset support.</li> <li>▶ MRU buffer pool algorithm.</li> </ul>
COPYTOCOPY	Additional recognized copies	N/A	N/A
DSN1COMP	N/A	<ul style="list-style-type: none"> <li>▶ Retrieve row "as is" instead of converting to current version.</li> <li>▶ New option to externalize dictionaries.</li> </ul>	N/A
DSN1COPY	N/A	Tolerates version system pages. Performance PTFS for page sets with CYL allocation.	N/A
DSN1LOGP	N/A	N/A	Message enhancement missing logs in range
DSN1PRNT	N/A	Recognizes a directory page and print in hex for format option.	N/A
LISTDEF	Dynamic definition of lists	N/A	N/A

Utility	DB2 V7	DB2 V8	DB2 9
LOAD	<ul style="list-style-type: none"> <li>▶ Family Cross Loader,</li> <li>▶ Online Load Resume and Partition Parallelism within a job.</li> </ul>	<ul style="list-style-type: none"> <li>▶ Delimited Input support.</li> <li>▶ Cross Loader support LOBS&gt;32 KB. LOB handling&gt;32 KB.</li> <li>▶ Allows &gt;254 compressed parts.</li> <li>▶ Performance PTFS. Permits use of ALIASes.</li> <li>▶ Auto invalidate for cached dynamic statements on completion.</li> <li>▶ UNIX System Services named Pipe support with templates.</li> <li>▶ CPU reductions.</li> </ul>	<ul style="list-style-type: none"> <li>▶ Copies an existing compression dictionary from the specified partition to other partitions.</li> <li>▶ Better information for replication like QReplication at utility termination.</li> <li>▶ Allows SORTKEYS NO.</li> </ul>
MODIFY RECOVERY	Performance enhancement	N/A	Modify Recovery new RETAIN® options (simplification and verification).
MODIFY STATISTICS	N/A	N/A	N/A
QUIESCE	N/A	N/A	N/A
Real Time Statistics	Introduced after DB2 V7 GA, including DSNACCOR	DSNACCOX.	Moved into DB2 catalog.
REBUILD	N/A	Online Schema support.	<ul style="list-style-type: none"> <li>▶ Rebuild Index SHRLEVEL CHANGE.</li> <li>▶ CPU reductions.</li> </ul>
RECOVER	N/A	N/A	<ul style="list-style-type: none"> <li>▶ RECOVER RESTOREBEFORE x'rbalrsn'.</li> <li>▶ Recovers to any point in time with consistency.</li> <li>▶ Supports large block interface for tapes.</li> <li>▶ RECOVER at object level from system level backups.</li> <li>▶ Displays progression of RECOVER log apply.</li> <li>▶ CPU reductions.</li> <li>▶ Better information for replication, such as Q replication provided by InfoSphere Replication Server product at utility termination.</li> </ul>

Utility	DB2 V7	DB2 V8	DB2 9
REORG	Fast switch, BUILD2 phase parallelism, and Drain and Retry	<ul style="list-style-type: none"> <li>▶ SORTKEYS, SORTDATA improvements.</li> <li>▶ SHRLEVEL NONE/REFERENCE REBALANCE. SHLRLEVEL CHANGE allows DISCARD.</li> <li>▶ SHRLEVEL REFERENCE catalog tables with links. ONLINE Schema support.</li> <li>▶ SCOPE PENDING support.</li> <li>▶ Automatic display claimers when a resource is unavailable. Reduces switch phase time on TS with COPY NO.</li> <li>▶ Switches to UTRW in UTILTERM for SHRLEVEL CHANGE.</li> <li>▶ Allows &gt;254 compressed parts. Performance PTFS.</li> <li>▶ Autoinvalidate for cached dynamic statements on completion.</li> <li>▶ Reset LPL and WEPR.</li> </ul>	<ul style="list-style-type: none"> <li>▶ BUILD 2 phase elimination.</li> <li>▶ Reorganizes LOB SHRLEVEL REFERENCE.</li> <li>▶ Partition Parallelism (UNLOAD/ RELOAD).</li> <li>▶ Parallel log applies.</li> <li>▶ Limit of 254 parts on compressed table space lifted.</li> <li>▶ Better information for replication, such as Q replication provided by the InfoSphere Replication Server product at utility termination.</li> </ul>
REPAIR	N/A	REPAIR versions for online schema support.	REPAIR LOCATE SHRLEVEL CHANGE.
RESTORE SYSTEM	N/A	New utility RESTORE SYSTEM.	<ul style="list-style-type: none"> <li>▶ Tape control added.</li> <li>▶ Support of incremental FlashCopy.</li> <li>▶ Restores from dump with parallelism.</li> </ul>
RUNSTATS	Statistics History	<ul style="list-style-type: none"> <li>▶ Updates statistics history tables without updating optimizer statistics.</li> <li>▶ Permit use of ALIASes.</li> <li>▶ HISTOGRAM statistics.</li> </ul>	Additional statistics.
TEMPLATE	Dynamic allocation of data sets	Supports UNLOAD/ LOAD from/ to HFS.	Template switching.
UNLOAD	To unload data from table spaces or copies	<ul style="list-style-type: none"> <li>▶ Delimited Data Support.</li> <li>▶ LOB handling&gt;32 KB. Permits use of ALIASes.</li> <li>▶ Performance PTFS.</li> <li>▶ UNIX System Services named pipe support with templates.</li> </ul>	<ul style="list-style-type: none"> <li>▶ Skip locked rows.</li> <li>▶ Unload from non-segmented image copy when table segmented.</li> </ul>

Other functions not included in the table, but worth mentioning, are the support for very large tables (DSSIZE) and the support for *pieces* for non-partitioning indexes that were introduced before DB2 Version 7.



## 1.4 Major changes since DB2 7

Since DB2 Version 7, DB2 introduced several new functions, as well as a different way of packaging the DB2 utilities.

### 1.4.1 Packaging of utilities

A basic set of core utilities have been included as part of DB2 since Version 1 was first delivered.

These utilities initially provided a basic level of services to allow customers to manage data. Some customers prefer to obtain such functions, however, from independent software vendors that have developed utility and tools offerings that offered additional performance, function, and features beyond that contained in the DB2 core utilities. With recent releases of DB2 for OS/390, in response to clear customer demand, IBM has invested in the improvement of the performance and functional characteristics of these utilities, as we have seen in the previous section.

With DB2 V7, most of the online utilities have been separated from the base product and they are now offered as separate products licensed under the IBM Program License Agreement (IPLA), and the optional associated agreements for Acquisition of Support. This combination of agreements provides DB2 users with benefits equivalent to the previous traditional ICA license.

Several utilities are included with DB2 at no extra charge. Other utilities are available as a separate product. The DB2 utilities are grouped as follows:

- ▶ The following utilities are core utilities, which are included (at no extra charge) with Version 9.1 of DB2 for z/OS:
  - CATENFM
  - CATMAINT
  - DIAGNOSE
  - LISTDEF
  - OPTIONS
  - QUIESCE
  - REPAIR
  - REPORT
  - TEMPLATE
  - All DSN stand-alone utilities
- ▶ All other utilities are available as a separate product called the DB2 Utilities Suite (5655-N97, FMIDs JDB991K), which includes the following utilities:
  - BACKUP SYSTEM
  - CHECK DATA
  - CHECK INDEX
  - CHECK LOB
  - COPY
  - COPYTOCOPY
  - EXEC SQL
  - LOAD
  - MERGECOPY
  - MODIFY RECOVERY
  - MODIFY STATISTICS
  - REBUILD INDEX
  - RECOVER

- REORG INDEX
- REORG TABLESPACE
- RESTORE SYSTEM
- RUNSTATS
- STOSPACE
- UNLOAD

All DB2 utilities operate on catalog, directory, and sample objects, without requiring any additional products.

The DB2 Utilities Suite is available as an optional product. You must separately order and purchase a license for these utilities.

### **DB2 Utilities Suite and DFSORT**

The DB2 Utilities Suite is designed to work with the DFSORT program, which you are licensed to use in support of the DB2 utilities even if you do not otherwise license DFSORT for general use. If your primary sort product is not DFSORT, consider the following informational APARs mandatory reading:

- ▶ APAR II14047/APAR II14213: Use of DFSORT by DB2 utilities
- ▶ II13495: How DFSORT takes advantage of 64-bit real architecture

These informational APARs are periodically updated.

We discuss the use of DFSORT in Chapter 6, “Sort processing” on page 141.

## **1.4.2 Functional enhancements**

In this section, we briefly describe the enhancements introduced since DB2 Version 7. For a detailed description of the functions, refer to the standard DB2 documentation or the IBM Redbooks publication *DB2 9 for z/OS Technical Overview*, SG24-7330 and *DB2 UDB for z/OS Version 8: Everything You Ever Wanted to Know, ... and More*, SG24-6079. For performance information, refer to the publication *DB2 9 for z/OS Performance Topics*, SG24-7473.

### ***DB2 utilities support for System z Integrated Information Processor (zIIP) and specialty engines***

In 2006, IBM introduced the zIIP to move eligible DRDA, selected DB2 utilities’ functions, and part of business intelligence (BI) workloads to a zIIP, to reduce software cost and improve the available capacity of existing general purpose engines. The DB2 utilities LOAD, REORG, and REBUILD INDEX now use the portion of the utility that is eligible to execute in a zIIP. The LOAD utility loads tables, and the REORG utility improves index performance, while the REBUILD INDEX utility creates or rebuilds your indexes. Only the build portion of the LOAD, REORG, and REBUILD DB2 utility processing is related to index maintenance and can be redirected to the zIIP.

Furthermore, DB2 utilities sorting fixed-length records in the memory object sort path is now eligible for being redirected to a zIIP. This enhancement potentially benefits almost all utility sort processing because, except for the data sort in REORG, currently all other sorts deal with fixed length records, even if the data is variable in length. DB2 pads out all indexes to the full length for the sort process. This processing is eligible for the additional offload. Only REORG data sorts handling variable length records are not eligible for the offload.

The expanded zIIP offload capability for sort requires changes to both DB2 and z/OS. You need::

- ▶ PTF UK48911 for DB2 Version 8 or PTF UK48912 for DB2 Version 9 for APAR PK85889, found at the following address:  
<http://www.ibm.com/support/docview.wss?uid=swg1PK85889>
- ▶ PTF UK48846 for z/OS V1.10 for APAR PK85856, found at the following address:  
<http://www.ibm.com/support/docview.wss?uid=isg1PK85856>

### ***Allocate the optimal number of data sets for use by DFSORT***

Utilities that invoke DFSORT often experience problems to allocate large sort work data sets through DFSORT. When utilities invoke DFSORT, they pass the SORTNUM value as a guidance about how many data sets should be used to allocate the required disk space. In constrained environments, DFSORT might not be able to allocate required disk space in the given number of data sets, causing a failure of this utility job.

DB2 utilities CHECK DATA, CHECK INDEX, CHECK LOB, LOAD, REBUILD INDEX, REORG TABLESPACE, and RUNSTATS have been enhanced to allocate sort work data sets dynamically before invoking DFSORT if the new DSNZPARM UTSORTAL is set to YES and no SORTNUM value is specified in the utility statement. Setting a new DSNZPARM IGNSORTN to YES will cause utilities to dynamically allocate sort work data sets even if the SORTNUM parameter was specified in the utility statement. The specified SORTNUM value will then be ignored. When the utility allocates sort work data sets dynamically, it will calculate the disk space requirements according to available record estimates and record length. To estimate the number of records to be sorted during utility execution, the processed object will be looked up in real-time statistics tables SYSIBM.SYSTABLESPACESTATS and SYSIBM.SYSINDEXSPACESTATS. If information is not available for the object (that is, TOTALROWS or TOTALENTRIES is set to NULL), the current estimation logic based on RUNSTATS catalog statistics will be used.

This function is described by and has been made available with the following maintenance:

- ▶ Informational APAR II14296, found at the following address:  
[http://www-01.ibm.com/support/docview.wss?rs=64&context=SSEPEK&q1=II14296&uid=sg1II14296&loc=en\\_US&cs=utf-8&lang=en](http://www-01.ibm.com/support/docview.wss?rs=64&context=SSEPEK&q1=II14296&uid=sg1II14296&loc=en_US&cs=utf-8&lang=en)
- ▶ PK45916 / UK33692 for DB2 Version 8, found at the following address:  
[http://www-01.ibm.com/support/docview.wss?rs=64&context=SSEPEK&q1=PK45916&uid=sg1PK45916&loc=en\\_US&cs=utf-8&lang=en](http://www-01.ibm.com/support/docview.wss?rs=64&context=SSEPEK&q1=PK45916&uid=sg1PK45916&loc=en_US&cs=utf-8&lang=en)
- ▶ PK41899/ UK33636 for DB2 Version 9, found at the following address:  
[http://www-01.ibm.com/support/docview.wss?rs=64&context=SSEPEK&q1=PK41899&uid=sg1PK41899&loc=en\\_US&cs=utf-8&lang=en](http://www-01.ibm.com/support/docview.wss?rs=64&context=SSEPEK&q1=PK41899&uid=sg1PK41899&loc=en_US&cs=utf-8&lang=en)

### ***BACKUP SYSTEM and RESTORE SYSTEM***

BACKUP and RESTORE SYSTEM utilities use disk volume FlashCopy backups and Copy Pool z/OS DFSMSHsm V1R5 constructs for system level point-in-time recovery.

BACKUP SYSTEM creates a nondisruptive fuzzy copy of all DB2 data, including catalog and directory.

RESTORE SYSTEM recovers the data using the database CopyPool as base and then it applies logs to the specified or truncation point.

In DB2 9, these utilities are enhanced to use new functions available with z/OS V1R8 DFSMSHsm and FlashCopy2. One enhancement allows *individual* table spaces or index spaces to be recovered. Maintaining multiple copies of all the data on disk can be expensive, DB2 9 also allows for the backup to be directed to tape. The I/O for physical copying to disk in the background can be reduced by the use of incremental FlashCopy. Even if incremental FlashCopy is used, the dump to tape is always a full dump.

### ***Increased availability with new online functions***

CHECK DATA, CHECK INDEX, and CHECK LOB is enhanced to run in a SHRLEVEL CHANGE mode. The V8 processing is the default or it can be specified as SHRLEVEL REFERENCE. SHRLEVEL CHANGE CHECK DATA, INDEX, and LOB work on a copy of the related table spaces and indexes. The copies (shadows) are taken by DB2 using the DFSMS ADRDSSU utility. Integrity checking is done on the shadows.

REBUILD INDEX can run now in SHRLEVEL CHANGE mode. The REBUILD utility will now have a LOGPHASE when SHRLEVEL CHANGE is specified. There are DRAIN WAIT options similar to REORG to control the final drain of writers before the index can be made available.

The REPAIR utility has been enhanced so that LOCATE can be run against indexes, index spaces, and table spaces with SHRLEVEL CHANGE. This does not apply to LOB table spaces.

### ***Template enhancements***

The new template switching function allows image copies of varying sizes to have different characteristics, for example, copy to disk if small or to tape if large. This provides significant flexibility in terms of the data set names and attributes, for example, device types. Also, the TEMPLATE utility was enhanced to support LOAD/UNLOAD from and to a z/OS UNIX file (HFS) or UNIX System Services named pipes.

### ***REORG enhancements***

The most important enhancement in REORG is the elimination of the BUILD2 Phase in Online REORG(OLR) maintaining the non-partitioned indexes (NPI). This enhancement has implications for REORG SHRLEVEL REFERENCE by part because it now rebuilds the whole of any NPI in full shadows. As the entire NPI is built and then switched, it is no longer possible to run reorganizations of different parts in separate jobs in parallel.

Furthermore, to reduce the elapsed time of the REORG utility for partitioned table spaces, REORG is now able to do the unload and reload of partitions in parallel. Note that this function applies to all types of SHRLEVEL; it is not confined to Online REORG (OLR). The SHRLEVEL CHANGE (OLR) has a further parallelism enhancement. REORG now attaches one or more subtasks during the LOG phase to speed up the processing of log records. In V8, for virtual storage reasons, REORG was limited to operating on 254 compressed partitions. This limit is now lifted in DB2 9. LOBs can be handled now in a similar way to regular table spaces with space reclamation, in that the data is unloaded and reloaded into a new data set. The processing of LOBs is controlled by the SHRLEVEL option. The options are now NONE and REFERENCE.

REORG automatically displays CLAIMERS when REORG receives a resource unavailable message and switches to UTRW during UTILTERM for REORG SHRLEVEL CHANGE. DB2 V8 introduced the rebalancing of partitions and discarding of rows with REORG SHRLVEL CHANGE.

### ***LOAD and UNLOAD enhancements***

The LOAD utility has been enhanced to accept data from a delimited file. The UNLOAD utility has also been enhanced to produce a delimited file when unloading the data. These enhancements help simplify the process of moving and migrating data into and out of DB2 for z/OS. Both LOAD and UNLOAD are enhanced to handle the floating decimal data type. UNLOAD has the ability to SKIP LOCKED DATA; the UNLOAD utility can skip rows on which incompatible locks are held by other transactions. To avoid the need for REORG to get compressed data LOAD COPYDICTIONARY allows priming of a partition with a compression dictionary.

LOAD and UNLOAD have been changed to allow LOBs to be loaded and unloaded into files (members in PDS, PDS/E, or HFS). The cross loader capability of LOAD has been enhanced for handling LOBs. Previously, the processing of LOB columns was limited to 32767 bytes, but is now restricted by memory above the 16 Mb line. In V8, there is an issue in that SORTKEYS became the default. SORTKEYS and LOB restart processing became incompatible. DB2 9 allows SORTKEYS to be turned off, by specifying SORTKEYS NO. DB2 will also allow RESTART(PHASE) for LOAD REPLACE of a table with a LOB.

### ***RECOVER enhancements***

In DB2 9 NFM, the RECOVER utility is enhanced to automatically detect the uncommitted work at the point in time selected for the recovery. DB2 will then roll back the changes on objects being recovered. After the recovery, all the objects involved in the recovery will be in a transactionally consistent state. During the LOGAPPLY phase of RECOVER, if the user issues the -DIS UTIL command, a new message DSNU116I will be issued. This new message indicates the range of the log that needs to be applied for the list of objects being recovered. It also shows the progress that has been made up to the last commit and the elapsed time since the start of the log apply phase of the recovery. The new keyword RESTOREBEFORE specifies that RECOVER is to search for an image copy, concurrent copy, or system-level backup with an RBA or LRSN value earlier than the specified X'byte-string' value to use in the RESTORE phase.

### ***Statistics enhancements***

For improved query optimization for non-uniform distribution, RUNSTATS can now collect the information by what are called *quantiles*. In these histogram statistics, RUNSTATS collects the frequency statistics of the distinct values of a column cardinality over the entire range, giving better selectivity and the potential for better access path selection.

To use the utilities effectively, you need to know when they need to be run. The new DB2 real-time statistics stored procedure (DSNACCOX) is a sample stored procedure that gives utility recommendations to help you maintain your DB2 databases.

### ***MODIFY RECOVERY enhancements***

A new option, RETAIN, can be used to retain a number of full image copies or even retrieve the number of image copies to retain from the GDGLIMIT into which the image copies were written.

### ***Stand-alone utilities enhancements***

DSN1COPY RESET is changed to reset the dictionary version to solve the copy from a non-data sharing DB2 subsystem to another non-data sharing DB2 subsystem problem reflecting the dictionary version of the source system.

DSN1COMP provides EXTNDICT functionality to externalize a compression dictionary created in DSN1COMP as an object module. In follow-on processing, this text deck can be used to generate a unique EDITPROC, which allows compression for DB2 data encrypted with the Data Encryption for IMS and DB2 Databases Tool.

DSN1LOGP has been enhanced to detect the situation when a range is specified for DSN1LOGP to print, but the entire range is no longer recorded in the BSDS because, for example, the archive logs have rolled off.



## Part 2

# Planning for DB2 utilities

In this part, we discuss how to plan for, and when to execute, the DB2 utilities. These considerations are discussed by exploiting utilities functions discussed in the following chapters:

- ▶ Chapter 2, “Managing partitions” on page 27
- ▶ Chapter 3, “Simplifying utilities with wildcarding and templates” on page 47
- ▶ Chapter 4, “Performance via parallelism of executions” on page 75
- ▶ Chapter 5, “Invoking and controlling executions” on page 93
- ▶ Chapter 6, “Sort processing” on page 141







## Managing partitions

Partitioning is the ability to physically “split” DB2 table spaces into multiple “parts” to allow the utilities and SQL to operate at a subset level of the table. This process greatly enhances the availability of DB2 objects. Partitioning also enables a total storage of data up to 128 TB; this size has steadily increased since V4, up to 131072 GB in DB2 9.

In this chapter, we examine the reasons for partitioning, and the best ways to manage partitions.

This chapter contains the following sections:

- ▶ Why partitioning
- ▶ Partitioning, non-partitioning, and data-partitioned secondary indexes and partitioned tables
- ▶ Universal table spaces
- ▶ Partition management
- ▶ SQL and partitioning

## 2.1 Why partitioning

Storing your DB2 data in partitions results in availability advantages, such as partition independence for DB2 utilities execution, for SQL, a better distribution of the data and therefore less contention, and for parallel access, better performance and reduced elapsed time. Using partitioned table spaces for all table spaces that are referred to in queries allows you to take advantage of query parallelism. Another reason for partitioning is to increase the amount of data that can be stored in a single (partitioned) table space. Utility jobs can be run at the partition level and operations on a table space can be broken along partition boundaries into jobs of more manageable size while allowing other applications to concurrently access data on other partitions. Additional advantages are if the data set backing a physical partition becomes damaged, the data outage is limited to that partition's data, and only that fraction of the data needs to be recovered. Furthermore, if partitioning is performed along application-meaningful lines, logical damage can be isolated to certain partitions, again limiting the data outage and recovery scope.

Refer to Chapter 1. "Table design" of *DB2 UDB for z/OS: Design Guidelines for High Performance and Availability*, SG23-7134 for more information about partitioning.

Table 2-1 shows how DB2 increased the maximum number of partitions in a partitioned table space from 254 to 4096 partitions since Version 7. As a result of the increased number of partitions in a partitioned table space, the maximum size of a partitioned table is increased from 16 terabytes to 128 terabytes.

Table 2-1 Number of partition and partitioned table space sizes

DB2 Version	Number of partitions	Maximum size	Total maximum size
V4 <sup>a</sup>	1 to 16	4 GB	64 GB
	17 to 32	2 GB	64 GB
	33 to 64	1 GB	64 GB
V5 <sup>b</sup>	254	4 GB	1,016 GB
V6 <sup>c</sup>	254	64 GB	16,256 GB
V7c	254	64 GB	16,256 GB
V8c	2048	64 GB	131,072 GB
V9c	2048	64 GB	131,072 GB

a. For a maximum total of 64 GB

b. Requires LARGE parameter in CREATE TABLESPACE

c. Requires DSSIZE parameter, DFSMS/MVS 1.5 and SMS managed table space

When defining a partitioned table space, DB2 usually distributes the data evenly across the partitions. If the distribution of the data becomes uneven, you can rebalance data among the partitions by redefining partition boundaries with no impact to availability. You can add a partition to the table and to each partitioned index on the table; the new partition becomes available immediately. A large table can be spread over several DB2 storage groups or data sets and the partitions must not belong to the same storage group. You can put the frequently accessed data in a partition of its own and on faster devices.

To avoid I/O contention among concurrently running jobs against the related non-partitioned indexes (NPI) for the partitioned table space, you can use the PIECESIZE parameter of the CREATE INDEX or the ALTER INDEX statement to modify the sizes of the index data sets. Because a partitioned table space contains more data sets than other table space types, more data sets will be opened by DB2 for data access. For extensive read-only queries, you can benefit from DB2 parallel processing. Spread the partitions over different disk volumes and allow each I/O stream to operate on a separate channel. Use the Parallel Sysplex® data sharing technology to process a single read-only query across many DB2 members in a data sharing group. Optimize Parallel Sysplex query processing by placing each DB2 member on a separate central processor complex. You must use EA-enabled (DFSMS function) table spaces or index spaces if you specify a maximum partition size (DSSIZE) that is larger than 4 GB in the CREATE TABLESPACE statement.

## 2.2 Partitioning, non-partitioning, and data-partitioned secondary indexes and partitioned tables

In this section, we highlight the difference between index-controlled partitioning and table controlled partitioning.

### 2.2.1 Index-controlled partitioning

Before DB2 V8, you only had *index-controlled partitioning* at your disposal to create a partitioned table. You created a table space specifying the NUMPARTS keyword and created the table to be placed in this table space. At that point, the definition was incomplete, because a partitioning index was required to indicate the values contained in each partition. When using index-controlled partitioning, concepts such as *partitioned*, *partitioning* and *clustering* are intertwined, because the index that defines the columns and key ranges for the different partitions is the partitioning index, is partitioned (made up of different physical partitions), and is the clustering index by default.

You define a partitioning index for a partitioned table space in order to complete the definition of the table space. The partitioning index specifies a partitioning key that dictates what columns the table is partitioned by, and one PART clause per partition to specify which rows (key ranges) go into which partition. The partitioning index also has to be the clustering index, so you must use the CLUSTER keyword or the statement fails. The partitioning index controls how the table is partitioned and how the table is clustered. This table and index combination is now called index-controlled partitioning.

Any index that does not specify the PART n VALUES keywords, and is defined on a partitioned table, is a *non-partitioned index* (NPI). NPIs cannot be physically partitioned, that is, they cannot have multiple physical partitions. They can only be allocated across multiple pieces to reduce I/O contention. A non-partitioned index may be encompassed by one physical data set or multiple data sets if piece size is specified. However, with *logical partitions*, all the keys of an NPI that point to rows of a physical data partition are considered to be a logical partition of the index. NPIs have a few operational drawbacks, such as recovery of a whole NPI, intersystem R/W requests in data sharing, difficult management of large objects, and so on.

For index-controlled partitioning, the limit keys are stored in both SYSIBM.SYSINDEXPART and SYSIBM.SYSTABLEPART in column LIMITKEY. Also, column IXNAME of SYSIBM.SYSTABLEPART contains the name of the index-controlled partitioning index. Indexes on index-controlled partitioned tables, and non-partitioned secondary indexes (non-partitioned and non-partitioning) on table-controlled partitioned tables, are identified with a value of '2' for the INDEXTYPE column of SYSINDEXES.

## 2.2.2 Table-controlled partitioning

DB2 V8 introduced *table-controlled partitioning* and a new type of index, the *data-partitioned secondary index* (DPSI). When using table-controlled partitioning, a table does not require a partitioning index (PI), as the partitioning is done based on the PARTITION BY clause in the CREATE TABLE statement. Figure 2-1 shows the differences between PI and DPSI.

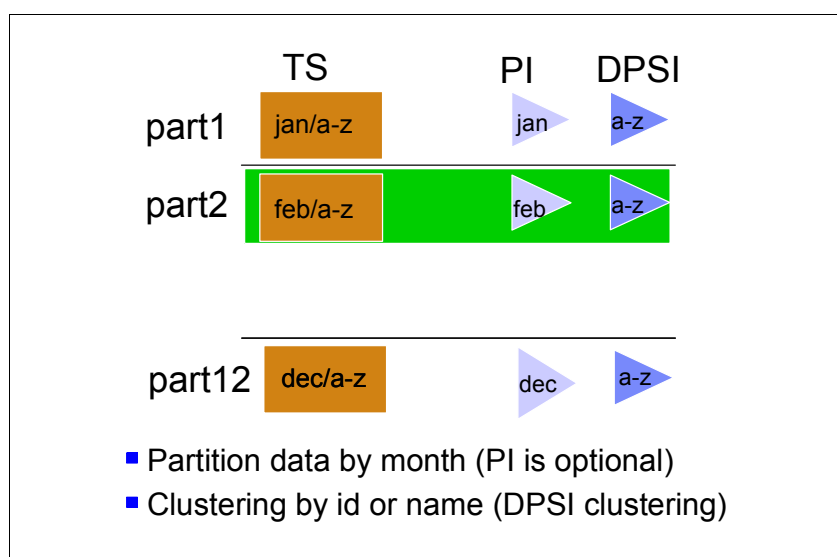


Figure 2-1 Partitioning index versus data-partitioned secondary index

Table-controlled partitioning is initiated by specifying the PARTITION BY clause on the CREATE TABLE statement. The PARTITION BY clause identifies the columns and values used to define the partition boundaries. When the new clause is used, the definition of the table is complete and data can be inserted into the table. Once table-controlled partitioning for a table is established, index-controlled partitioning is no longer an option for that table. Any attempt to create an index on this table with the VALUES or ENDING AT keywords is disallowed. The table is complete after executing the CREATE TABLE statement; no partitioning index is required.

Example 2-1 shows how to create a table controlled partitioned table. If desired, these partition boundaries can be added by specifying ALTER TABLE with the ADD PARTITION BY RANGE clause after the table space and table have been created. For table-controlled partitioning, the limit keys are only stored in columns LIMITKEY and LIMITKEY\_INTERNAL of SYSIBM.SYSTABLEPART. A value of 'P' for the INDEXTYPE column of SYSINDEXES indicates that an index is both partitioned and partitioning.

---

*Example 2-1 Create a table controlled partitioned table*

---

```
CREATE TABLE TRANS
(ACCTID ...,
STATE ...,
POSTED ...,
... , ...)
PARTITION BY (POSTED)
(PARTITION 1 ENDING AT ('01/31/2003'),
PARTITION 2 ENDING AT ('02/28/2003'),
...
PARTITION 13 ENDING AT ('01/31/2004'));
```

---

*Data-partitioned secondary indexes* can improve data availability during utility operations that operate at the partition level, such as REORG PART, LOAD PART, CHECK PART, REBUILD INDEX PART, RUNSTATS PART, and RECOVER DSNUM. DPSIs allow total concurrent operations. In prior DB2 versions, all secondary indexes were non-partitioned; only the partitioning index could be partitioned. A partitioned index is made up of multiple physical partitions, with one per data partition.

The index keys in each index partition correspond to the rows in the same data partition number, that is, index partition 1 only contains keys for those rows found in data partition 1, index partition 2 only contains keys for those rows found in data partition 2, and so on. A partitioned index has the keyword PARTITIONED specified in the CREATE INDEX statement that defines it. A DPSI is made up of multiple physical partitions (that match the partitioning scheme of the table). It is an index that is partitioned based on data rows. A DPSI contains different columns than the partitioning columns or in a different order or collating sequence than those that partition the table. DPSIs must allow duplicates and must not be unique. DPSIs are identified with a value of 'D' for the INDEXTYPE column of SYSINDEXES.

DPSI's organization promotes high data availability by facilitating efficient utility processing on data partitioned secondary indexes. It streamlines partition-level operations such as adding and rotating partitions, also introduced in DB2 V8. Because keys for a given data partition reside in a single DPSI partition, partition-level operations can take place at a physical versus logical level. Thus, partition-level operations are facilitated to the extent that a table's indexes are partitioned. For Online Reorg (OLR), there is no BUILD2 phase processing for DPSIs. Because keys for a given data partition reside in a single DPSI partition, a simple substitution of the index partition newly built by REORG for the old partition is all that is needed. If all indexes on a table are partitioned (partitioned PI or DPSIs), the BUILD2 phase of REORG is eliminated in DB2 V8.

In DB2 9 for z/OS, the BUILD2 phase has been eliminated to improve availability of your data. For LOAD there is no contention between LOAD PART jobs during DPSI processing. This is because there are no shared pages between partitions on which to contend. Thus, if all indexes on a table are partitioned, index page contention is eliminated. During parallel LOAD PART job execution, each LOAD job inserts DPSI keys into a separate index structure, in key order. This allows the LOAD utility logic to follow an efficient append strategy (instead of following a *row at a time* logic). Data partitioned secondary indexes also improve the recovery characteristics of your system. DPSIs can be copied and recovered at the partition level. Individual partitions can be rebuilt in parallel to achieve a fast rebuild of the entire index.

Figure 2-2 shows how to create a DPSI and the difference from a non-partitioned secondary index (NPSI).

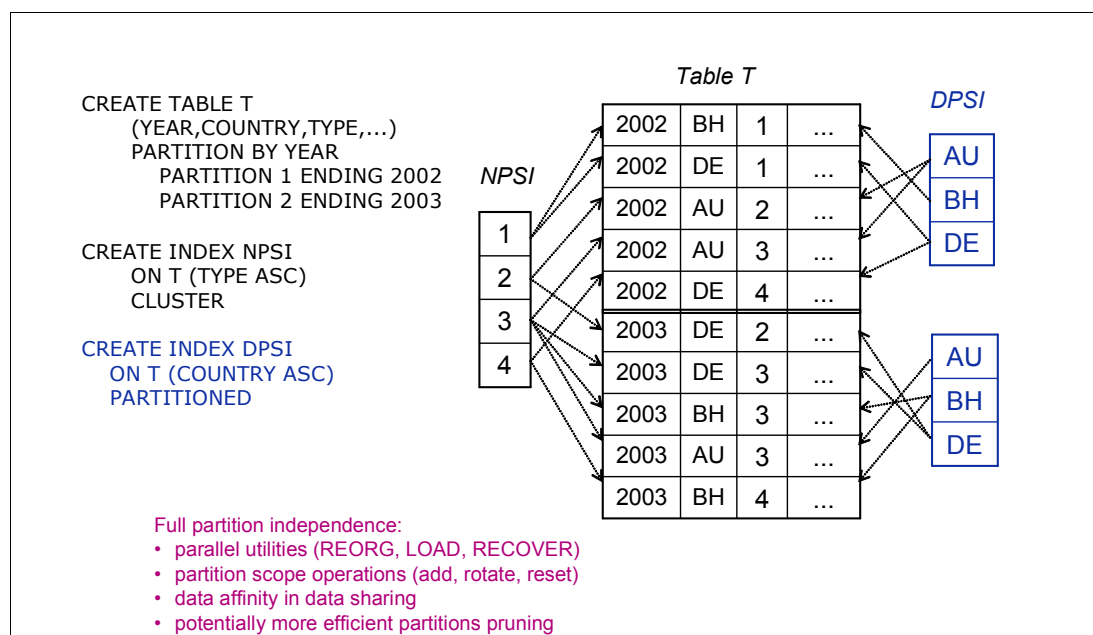


Figure 2-2 DPSI versus NPIs

## 2.3 Universal table spaces

DB2 9 introduced a new type of table space: a universal table space (UTS). A universal table space is a table space that combines the characteristics of both a segmented and a partitioned table space. Two types of universal table spaces are available: the partition-by-growth table space and the range-partitioned table space. Table spaces that are both segmented and partitioned are defined using both the **SEGSIZE** and **NUMPARTS** parameters. The type of universal table space is determined depending on the specification of the **SEGSIZE**, **MAXPARTITIONS**, or **NUMPARTS** clauses. If you specify **SEGSIZE** with **NUMPARTS**, the table space is created as a range-partitioned universal table space. If **SEGSIZE** is specified with **MAXPARTITIONS**, the table space is created as a partition-by-growth universal table space.

Table 2-2 shows the specifications by table space type.

Table 2-2 Table space type by **SEGSIZE**, **MAXPARTITIONS**, and **NUMPARTS** values

<b>SEGSIZE</b> clause	<b>MAX PARTITIONS</b> clause	<b>NUMPARTS</b> clause	Type of table space
Specified	Specified	Not specified	Partition-by-growth table space
Specified	Not specified	Not specified	Segmented table space
Specified	Not specified	Specified	Range-partitioned universal table space

SEGSIZE clause	MAX PARTITIONS clause	NUMPARTS clause	Type of table space
Not specified	Specified	Not specified	Partition-by-growth table space with an implicit specification of SEGSIZE 4
Not specified	Not specified	Not specified	Segmented table space with an implicit specification of SEGSIZE 4
Not specified	Not specified	Specified	Partitioned table space

APAR PK85881 introduces a DSNZPARM setting that controls whether a UTS is created using Basic Row Format (BRF) or Reordered Row Format (RRF)<sup>1</sup>.

### 2.3.1 Partition-by-growth table spaces

Partition-by-growth (PBG) table spaces are useful for those table spaces where the tables do not have a suitable partitioning key, but are expected to exceed the 64 GB limit for simple or segmented table spaces up to DB2 V8. A partition-by-growth table space has a limit of 128 TB. A partition-by-growth table space starts with a single-partition table space and automatically grows additional partitions as needed. When the first partition fills, you will not receive a resource unavailable condition; rather, the next partition is created, and so on. This continues up to a maximum number of partitions that you define at the creation time of the table space. If your table space finally reaches this maximum value, you will then receive the resource unavailable condition. The data structure of partition-by-growth table spaces is similar to the structure of a segmented table space.

This structure has the advantage of better space management and mass delete capabilities against a simple table space or a regular partitioned table space. Because partition-by-growth table spaces are segment partitioned, they offer almost all capabilities of partition-level operation and parallelism. However, one exception is the LOAD utility that does not allow LOAD PART for partition-by-growth table spaces. With the new CREATE/ ALTER TABLESPACE keyword MAXPARTITIONS, you specify the maximum number of partitions to which the partition-by-growth table space may grow. For an implicitly created table space, you can influence the number of partitions by using the PARTITION BY SIZE clause on your CREATE TABLE statement. Example 2-2 shows the SQL CREATE TABLESPACE statement for PBG.

*Example 2-2 Partition-by-growth CREATE TABLESPACE statement*

---

```
CREATE TABLESPACE TS1 IN DB1
  MAXPARTITIONS 55
  SEGSIZE 64
  DSSIZE 2G
  LOCKSIZE ANY;
```

---

<sup>1</sup> Refer to 7.2, “Loading into reordered row format” on page 176.

Example 2-3 shows the SQL CREATE TABLE statement for PBG. Keep in mind that it is only available when you do not specify a table space name in the CREATE TABLE statement. The table space is implicitly created. The *integer G* specifies the DSSIZE of the table space.

Example 2-3 Partition-by-growth CREATE Table statement

```
CREATE TABLE MYTABLE
PARTITION BY SIZE
EVERY INTEGER G;
```

Figure 2-3 gives an overview of the DB2 utilities and their concurrency to PBG table spaces.

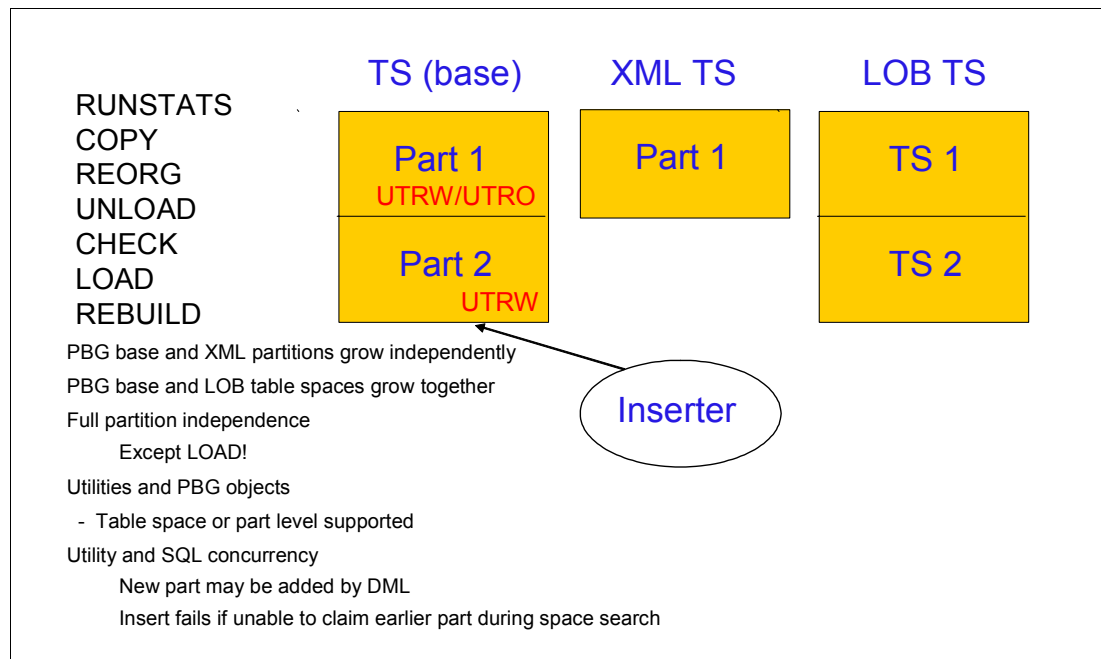


Figure 2-3 DB2 utilities and PBG concurrency

## Involved catalog tables

As with many new functions, the catalog tables have been adjusted to reflect their usage and object characteristics.

### ► SYSIBM.SYSTABLESPACE

A new column MAXPARTITIONS has been added to SYSIBM.SYSTABLESPACE. Its entry is the number that you have specified in your CREATE or ALTER TABLESPACE statement. It does not show the actual number of physically existing partitions. The actual number of allocated partitions is shown in column PARTITIONS.

Column TYPE shows a value of 'G' for partition-by-growth table spaces.

**Note:** Even though the number of MAXPARTITIONS might be, for example, 100, and the number shown in PARTITIONS is only 4, the data is only spread over four physical partitions. It is not possible to use ALTER TABLESPACE to reduce MAXPARTITIONS after the table space has been created.



► **SYSIBM.SYSTABLEPART**

When you create the table space, only one row is added to SYSIBM.SYSTABLEPART independently from the number that you used for MAXPARTITIONS. Additional rows are added to this catalog table as your table grows and additional partitions are allocated because your amount of data has exceeded the associated DSSIZE.

## Utility considerations

The following sections describe the different considerations regarding PBGs when you deal with the various DB2 utilities.

### DBET states

Figure 2-4 shows how DBET states are inherited when dealing with PBGs.

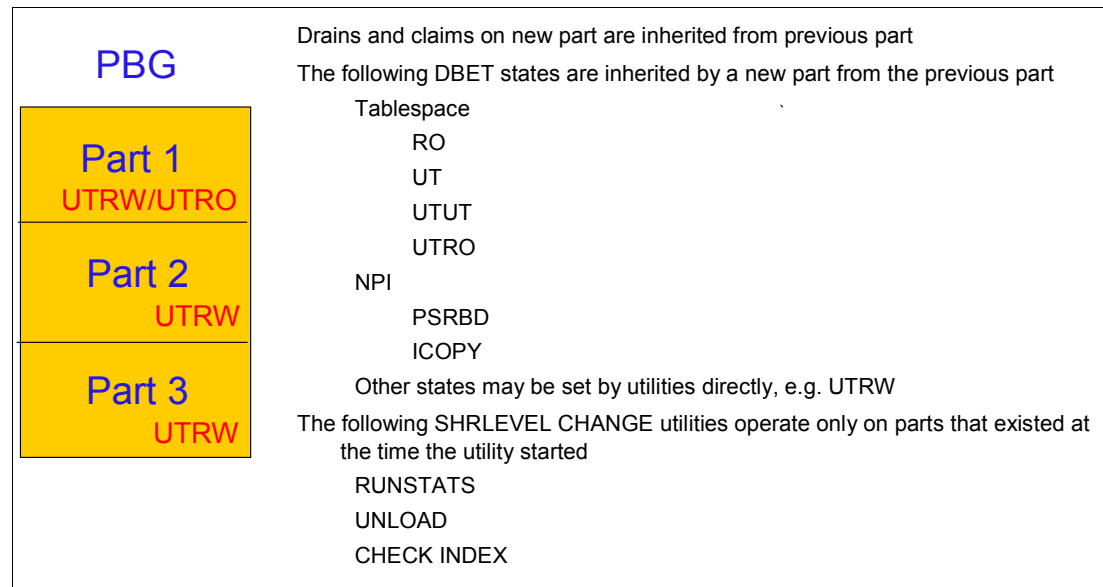


Figure 2-4 Utilities and PBGs

### COPY

An image copy at the table space level with SHRLEVEL(CHANGE) contains new partitions added by SQL INSERTS after the image copy began. The newly added partitions are recoverable via the DB2 logs. The COPY utility allows you to copy either the entire table space or perform the copy on a partition basis. When you make an image copy of a partition-by-growth table space and partitions are empty as a result of REORG, SQL delete operations, and so on, the COPY utility still copies the empty partition because the empty partition has a header, space map pages, or system pages. You can:

- Copy at the part or table space level
- Copy new parts if there is a SHRLEVEL CHANGE at the table space level

### LISTDEF

With the PARTLEVEL keyword, you will get an entry for each partition that exists when the LISTDEF list is evaluated. Partitions that are added after the list is evaluated will not be in the list. Long-running job steps in which the list is reused and partitions were added during the job step will not process the new partitions. Restarted jobs use the original list saved during the original execution. A restarted utility job using a PARTLEVEL list is running with the original list that was saved during the original execution for a later restart. The list will not include the added partitions.

## **LOAD**

The PART option that specifies that data is to be loaded into a partition of a partitioned table space is not valid for partition-by-growth table spaces and there is no partition parallelism. If you are using LOAD for a partition-by-growth table space, you can load data only at the table space level, not at the partition level. If additional partitions during the LOAD process are needed and the maximum number of partitions for the table space is not reached, the LOAD utility will automatically trigger the process to add partitions. The LOAD fails when the maximum number of partitions is reached. LOAD builds only one compression dictionary that is populated through all partitions. If all rows of a partition are deleted, the partition is kept as empty.

## **REBUILD INDEX**

The REBUILD INDEX utility may reset more partitions than it repopulates. Excess index partitions continue to exist as empty partitions and will not be removed.

REBUILD INDEX SHRLEVEL CHANGE index entries for new parts are included in the log apply phase.

## **RECOVER**

Adding a new partition to an partition-by-growth table space establishes a recoverable point. When recovering a partition-by-growth table space to a point in time (PIT) that has an image copy with fewer partitions than the current table space, any excess partitions (currently defined partitions that are not all in the image copy) will be empty after the RECOVER processing. Once a partition is added and its associated data set is allocated, DB2 will not roll back this allocation even if the entire Unit of Recovery (UR) that caused the addition of the new partition is rolled back. If the UR that would cause DB2 to add a new partition contains an outstanding DDL operation in the same UR, DB2 does not allocate the new partition and rolls back the whole UR instead.

## **REORG**

The REORG TABLESPACE utility condenses the data into the minimum number of required partitions by eliminating existing holes. The exception is LOBs; if the table space contains LOB columns, the data will be not moved from one partition to another.

Because the REORG TABLESPACE utility cannot reclaim physical space, if the data can fit into two partitions after a REORG, the third partition is not deleted.

If additional space is required, the REORG TABLESPACE utility triggers the process to add additional partitions and, if compressed, the dictionary is copied from the previous part. If the maximum number of partitions has been reached, REORG fails.

REORG TABLESPACE PART fails if the data does not fit back into its partition because of the change in the free space parameter during the REORG. To prevent the utility from failing, run REORG TABLESPACE on the entire table space or modify the free space parameter to fit the data rows into the partition.

REBALANCE for partition-by-growth table spaces is not supported and you cannot use REORG to reclaim the space for dropped tables.

REORG of part range can move the data from one part to another within range, except when LOB columns exist.

For REORG with SHRLEVEL CHANGE, a new part is added to both base and shadow tables space. Parallelism and REBALANCE are not allowed.

### ***DSN1COMP***

The DSN1COMP utility has the option to determine compression saving estimates at the data set level for a single partition.

### ***DSN1COPY***

When you use DSN1COPY on a partition-by-range table space to copy a partition-by-range table space, use the SEGMENT and Numparts options. For partition-by-growth table spaces, the Numparts value specified should be the MAXPARTITIONS value with which the table space was created.

If you use the DSN1COPY utility to restore a partition or an entire table space for a partition-by-growth table space, the total number of partitions in a DSN1COPY might not be consistent with the number of partitions defined on the current table space. To avoid residual data, delete data in the excess partitions from the table space before you apply the DSN1COPY utility.

DSN1COPY can only be used if the number of active partitions of the source and the target table space are the same; otherwise, you should use the UNLOAD and the LOAD utilities to unload and reload the data into the target table space. All the target data sets must exist. You can use Access Method Services to define them.

### **Considerations**

Keep the following considerations in mind:

- ▶ Partition-by-growth table spaces must be storage group controlled.
- ▶ The MEMBER CLUSTER option of the CREATE TABLESPACE syntax is not valid.
- ▶ You cannot use the following options for partition-by-growth table spaces:
  - ALTER TABLE ADD PARTITION
  - ALTER TABLE ROTATE PARTITION
  - ALTER TABLE ALTER PARTITION
- ▶ LOB and XML spaces associated with a partition-by-growth table space are always implicitly defined by DB2, independent of whether SQLRULES DB2 or STD is in effect.
- ▶ A partition-by-growth table space can only contain one table.
- ▶ Only NPIs are allowed on tables residing in partition-by-growth table spaces.
- ▶ When a new partition is added, drain and claim values are inherited from a prior partition.
- ▶ Some DBET states are also inherited from the previous partition when a new partition is added. Those DBET states include RO\*, UTUT, UTRO, PSRBD, and ICOPY.

## **2.3.2 Range-partitioned universal table space**

Range-partitioned (partitioned by range or PBR) universal table spaces is the other type of UTS that combines partitioned and segmented table spaces. You create this type of table space when you specify both keywords Numparts and SEGsize in one CREATE TABLESPACE statement. Once you have created a range-partitioned table space, you can take most of the same actions that used to be allowed on partitioned or segmented table spaces. As for regular partitioned table spaces, only one table is allowed.

Some benefits of range-partitioned table spaces are:

- ▶ Better space management in conjunction with varying-length rows, because a segmented space map page has more information about free space than a partitioned space map page.
- ▶ Improved mass delete performance. Mass delete in a segmented table space organization tends to be faster than in other types of table space organizations, such as partitioned or simple table spaces.
- ▶ DB2 reuses all or most of the segments of a table after you committed the dropping or mass deletion of it.

Example 2-4 shows the SQL CREATE statement for a PBR. The SEGSIZE and Numparts clauses define the PBR. Create a partitioned table space and just add the SEGSIZE and Numparts clauses. Then create a table-controlled partitioning table.

**Attention:** No index-controlled partitioning is allowed. If you attempt this action, you will receive the SQLCODE = -662 error.

---

*Example 2-4 Create Partition By Range*

```
CREATE TABLESPACE PRB_TS1 IN UTS_DB1
  Numparts 3
  SEGSIZE 64
  LOCKSIZE ANY;
CREATE TABLE MyTable
  ( C1 CHAR(4),
    C2 VARCHAR(20),
    C3 INTEGER )
  PARTITION BY (C1)
  ( PARTITION 1 ENDING AT ('DDDD'),
    PARTITION 2 ENDING AT ('HHHH'),
    PARTITION 3 ENDING AT ('ZZZZ') )
  IN UTS_DB1.PRIB_TS1;
```

---

## Miscellaneous considerations

Keep the following considerations in mind;

- ▶ All range-partitioned universal table spaces are LARGE table spaces.
- ▶ Column TYPE in SYSIBM.SYSTABLESPACE contains an 'R' for range-partitioned universal table spaces.
- ▶ The MEMBER CLUSTER keyword is not allowed for range-partitioned universal table spaces.
- ▶ Range partitioned table spaces follow the regular partitioned table space locking scheme (that is, LOCKSIZE TABLE is not a valid keyword in the CREATE TABLESPACE syntax).
- ▶ You need to use the table-controlled partitioning syntax introduced with V8 to create range-partitioned universal table spaces.
- ▶ Range-partitioned universal table spaces exist in addition to the regular partitioned table space.
- ▶ Mass delete locks are at the table level instead of the table space level.
- ▶ If a range-partitioned universal table space contains an XML column, the corresponding XML table space will be range-partitioned universal as well.

## 2.4 Partition management

DB2 has the ability to immediately add partitions, rotate partitions, change the partitioning key values for table-controlled partitioned tables via the ALTER TABLE statement, and rebalance partitions. Here we give a brief description of these functions.

### 2.4.1 Adding a partition to an existing regular-partitioned or range-partitioned universal table space

With DB2, it is easy to add partitions at a later date. Therefore, you can now start out with a limited number of partitions in your applications, as many as you currently need, and then reevaluate your partition needs when needed. You can use the ALTER TABLE statement to add a partition to an existing regular-partitioned or range-partitioned universal table space and to each partitioned index in the table space. You do not specify a partition number on the ALTER TABLE statement, you only specify a new limit key for the new partition. You can specify space attributes for a new partition by using the space attributes of the ALTER TABLESPACE and ALTER INDEX statements. Partitions will be added up to the maximum limit, which is determined by the parameters specified when the partitioned table was initially created. To add a partition, issue a statement similar to this one:

```
ALTER TABLE TRANS ADD PARTITION ENDING AT ('12/31/2004');
```

When you add a partition, DB2 uses the next physical partition that is not already in use until you reach the maximum number of partitions for the table space. When DB2 manages your data sets, the next available data set is allocated. When you manage your own data sets, you must first define the data sets for the table space and the partitioned indexes before adding a new partition. For the table and index space, DB2 uses the existing table/ index space PRIQTY and SECQTY attributes of the previous partition and index partition. If the previous partition is in REORP status, the new one will be too. After adding the new partition, the table space is immediately available.

Auto rebinds will occur, because if a partition is added or one of the existing partitions is changed or rotated, all the plans, packages, and dynamic cached statements that refer to the table are invalidated. This is a necessary action, because the access path is optimized to read only certain partitions. You can start inserting or loading rows into the new partition immediately if the table space is a large table space (table-partitioned) because the high-limit key is always enforced. For non-large table spaces (index-partitioned), the partition is placed in a REORP state, because the last partition boundary was not previously enforced and the high-limit key is not enforced. A recoverable point will be established when a table has the NOT LOGGED attribute and an ALTER TABLE with the ADD PARTITION clause is executed.

Other miscellaneous considerations for ADD PARTITION are:

- ▶ ALTER PARTITION is not allowed for an existing partitioned table space if the table has LOB columns.
- ▶ You cannot add or alter a partition for a materialized query table.
- ▶ You cannot explicitly add a new partition to a table created in the partition-by-growth table space.
- ▶ If you do a PIT recovery prior to the addition of a partition, DB2 cannot roll back the definition of the partition. DB2 clears all data from the partition, and the partition remains part of the database.

## 2.4.2 Rotate partition

Rotating partitions allows old data to “roll off” while reusing the partition for new data by using the ALTER TABLE ROTATE PARTITION FIRST TO LAST statement. A typical case is where 13 partitions are used to continuously keep the last 12 months of data. When rotating, you can specify that all the data rows in the oldest (or logically first) partition should be deleted, and then specify a new table space high boundary (limit key) so that the partition essentially becomes the last logical partition in the sequence, and is ready to hold the data that is added. The lowest logical partition is assigned to become the next logical partition.

To rotate the first partition to be the last partition, issue an ALTER TABLE table-name statement with the ROTATE PARTITION option. The new partitioning key value in the ENDING AT clause must be higher than the current high keylimit if partition values are ascending. If partition values are descending, the new key limit must be lower than the current low key limit. To rotate partitions, issue a statement like the following:

```
ALTER TABLE TRANS ROTATE PARTITION FIRST TO LAST ENDING AT ('12/31/2008') RESET;
```

A referential constraint with DELETE RESTRICT on the table does not allow the ROTATE operation. Index-controlled partitioned table spaces are converted to table-controlled partitioning. The RESET keyword in the ALTER TABLE statement specifies that the existing data in the oldest partition is deleted, and no delete triggers are activated. After the ALTER operation completes, the partition is not placed in REORP status and it is immediately available. If the previous partition is in REORP status, the new partition will inherit that status and will be in REORP as well. You have to run a REORG before both partitions can be used by our applications. After a rotate operation, if the partitioning key is ascending, DB2 prevents an INSERT of a row with a null value for the key column. If the partitioning key is descending, DB2 allows an INSERT of a row with a null value for the key column.

The row is inserted into the first partition. After using the ALTER TABLE ROTATE PARTITION statement, the logical and physical order of the partitions is no longer the same. The display command lists the status of table space partitions in logical partition order. Logical order is helpful when investigating ranges of partitions that are in REORP. It enables one to more easily viewed groupings of adjacent partitions that may be good candidates for reorganization. When used in conjunction with the SCOPE PENDING keyword REORG, a reasonable subset of partitions can be identified if one wants to reorganize REORP ranges in separate jobs. The logical partition number can also be found in the catalog in several places. Both SYSTABLEPART and SYSCOPY have a LOGICAL\_PART column.

Other miscellaneous considerations about ROTATE PARTITIONS are:

- ▶ ROTATE PARTITION is not allowed for a table in a partition-by-growth table space or a table that has XML columns.
- ▶ Using the RESET keyword DB2 will delete all the rows of the existing partition that you are about to reuse. During the rotate process, a DBD lock is held. However during a regular deletion, there is additional impact on the logging. To speed up the deletion process, you can use LOAD REPLACE with an empty data set before running the ALTER TABLE... ROTATE PARTITION statement. The reset operation requires that the keys for the rows of that partition must also be deleted from all non-partitioned indexes. Each NPI must be scanned to delete these keys; therefore, the process can take an extended amount of time to complete, as each NPI is processed serially.
- ▶ Rotating a partition occurs immediately. If there is a referential constraint with DELETE RESTRICT on the table, the ROTATE might fail. If the table uses index-controlled partitioning, it is converted to use table-controlled partitioning.

- ▶ You can recover the partition to the current time or to a point in time after the ALTER operation. The RECOVER utility can use a recovery base (for example, a full image copy, a REORG LOG YES, or a LOAD REPLACE LOG YES operation) that occurred prior to the ALTER operation. You cannot recover the partition to a point in time prior to the ALTER; the recover fails with messages MSGDSNU556I and RC8.
- ▶ You cannot recover a table and index space to a point in time that is prior to rotating partitions. After you rotate a partition, you cannot recover the contents of that partition. All SYSCOPY and SYSLGRNX entries are deleted when the rotate statement is executed, and DB2 does not support an “undo” of the rotate statement.
- ▶ A REORG REBALANCE might not be possible if the logical and physical partition numbers for the specified table space do not match. This situation can be created by a series of ALTER ROTATEs and ALTER ADD PARTs.
- ▶ After an ALTER TABLE ROTATE PARTITION process finishes, run RUNSTATS with REORG to ensure a effective access path.

### 2.4.3 Rebalance partitions dynamically

For index established partitions for the table space, you can use the ALTER INDEX index-name PART x VALUES ('new-limit-key') to alter the partition boundaries of a partitioned table space. This puts the affected partition and the next partition in the REORG pending (REORP) state. A REORG job redistributes the rows between both partitions according to the new limit key.

Starting with DB2 V8, you can use the REBALANCE option of the REORG utility to rebalance partitions. The option works for index-controlled and table-controlled partitioned table spaces. The purpose of REBALANCE with REORG TABLESPACE is to set new partition boundaries so that all the rows participating in the reorganization are evenly distributed across the partitions being reorganized. The SYSTABLEPART and SYSINDEXPART tables are updated during this process so that they contain the new limit key values. When doing a REBALANCE REORG, the data is unloaded and sorted by the partitioning column(s) of the partitioned table space. Then REORG TABLESPACE with REBALANCE calculates the approximate number of rows expected to be populated across all the partitions, taking into account the percentage of free space allowed on each page, as well as the free pages specified (can differ between partitions). When reloading the data, REORG TABLESPACE notes the value of the partitioning columns as each partition reaches the row count threshold.

At the end of a successful REORG, the catalog and directory are updated with the new partition boundaries. You can specify REBALANCE with SHRLEVEL NONE or SHRLEVEL REFERENCE. REBALANCE cannot be specified with SHRLEVEL CHANGE or SCOPE PENDING.

When the clustering index does not match the partitioning key, REORG must be run twice on the partition range to ensure that the rows are in optimal clustering order. The first reorganization (with the REBALANCE option) moves data rows to the appropriate partition. After the first reorganization, the table space is put into an AREO\* state to indicate that another reorganization is recommended. The second reorganization (without the REBALANCE keyword) orders each data row in clustering order (based on the clustering index) within the appropriate partition.

When running a REORG REBALANCE, you must create a inline copy. However, at REBALANCE completion, DB2 invalidates plans, packages, and the dynamic cache.

Other miscellaneous considerations on REBALANCE:

- ▶ You cannot use the REBALANCE option for partitioned-by-growth table spaces, base tables with XML columns, or XML table spaces.
- ▶ You cannot specify the REBALANCE option for partitioned table spaces with LOB columns.
- ▶ You cannot specify the keyword REBALANCE together with SHRLEVEL CHANGE, or with the SCOPE PENDING, OFFPOSLIMIT, INDREFLIMIT, REPORTONLY, UNLOAD ONLY, and UNLOAD EXTERNAL keywords.
- ▶ Perfect rebalancing is not always possible if the columns used in defining the partition boundaries have many duplicate values within the data row. As all keys with a certain value have to go into the same partition, a key with many duplicates can lead to a partition that is bigger than the other partitions or cause one or more partitions to have no rows.
- ▶ REBALANCE cannot be specified for an object that is involved in a clone relationship. Because the base and clone tables share catalog information, this can change the partition boundaries of the target table.
- ▶ You cannot use unload parallelism with REORG REBALANCE.
- ▶ If you change partition boundaries with ALTER or REORG REBALANCE, you can recover the partition to the current time if a recovery base (for example, a full image copy, a REORG LOG YES operation, or a LOAD REPLACE LOG YES operation) exists. You can recover the partition to a point in time after the ALTER, or recover the partitions that are affected by the boundary change to a point in time prior to the ALTER; RECOVER sets REORG-pending status on the affected partitions and you must reorganize the table space or range of partitions. All affected partitions must be in the recovery list of a single RECOVER statement.

**Note:** The partition range that you specify on the REORG REBALANCE statement is a range of physical partitions. When rebalancing, DB2 unloads, sorts, and reloads the data based on logical partition numbers. If, because of earlier partition rotations or adding additional partitions, logical and physical partitions no longer match up, REORG REBALANCE may not be possible. When reorganizing, the physical partition numbers associated with the logical partition number that DB2 uses to reload data into must be within the physical partition range that you specify on the REORG REBALANCE statement; otherwise, you receive a DSNU1129I message, and REORG terminates.

## 2.4.4 ALTER partition boundary

You can modify the limit keys for table-based partitioning table partitions, for example, with the following SQL statement:

```
ALTER TABLE. ... ALTER PARTITION 60 ENDING AT ('01/01/2010');
```



For any affected partition except for the last, both the identified partition and the partition that follows are placed in REORG-pending (REORP) status. Any ALTER related to a partitioned table has to specify the physical partition number. You must derive the physical partition number from the logical partition number. SYSIBM.SYSTABLEPART and SYSIBM.SYSCOPY have a LOGICAL\_PART column to assist you with the conversion between a logical and physical partition number. If the table uses index-controlled partitioning, it is converted to use table-controlled partitioning. The high limit key for the last partition is set to the highest possible value for ascending key columns or the lowest possible value for descending key columns.

### Implications for PIT recovery

If recovery to a point in time prior to the ALTER of partition boundaries is requested, then the RECOVER statement should include all affected partitions. If only a subset of affected partitions are listed, RECOVER will fail with RC=8 to ensure that all partitions affected are listed in the RECOVER utility statement.

At the end of the successful recovery, the partitions will be in a REORP state. RECOVER to a point in time does not roll back the ALTER of the partition boundaries, so the partitions must be reorganized after the recovery to place the data back into the correct partitions.

### Changing the partition boundaries with REORG REBALANCE

It is possible to change partition boundaries with the following SQL statements:

```
ALTER TABLE table-name ALTER PARTITION integer ENDING AT (constant)
```

when you use table controlled partitioning, or

```
ALTER INDEX index-name ALTER PARTITION integer ENDING AT (constant)
```

when you use index-controlled partitioned table spaces.

When altering the partition boundaries as above, it is your responsibility to determine the new partition boundaries. In both cases, if data exists in the partition where the range was decreased, the partitions on both sides of the boundary are placed in REORP status.

Alternatively, you can rebalance the data in your partitions by using the REBALANCE option of the REORG utility as follows:

```
REORG TABLESPACE dbname.tsname PART(n:m) REBALANCE
```

This method avoids putting the partitions in a REORP state and making the data unavailable for applications until the REORG has completed. When you use REORG SHRLEVEL REFERENCE to rebalance the partitions, the data is available for readers almost all of the time. Only during the switch phase is the data unavailable. It is also during the switch phase that DB2 updates the limit keys to reflect the new partition boundaries.

## 2.5 SQL and partitioning

We have briefly explained some of the benefits to SQL processing from partitioning, that is, less data in a partition, fewer rows to read or write in parallel, and more I/O performed in a unit of time, which leads to shorter elapsed time.

Partitioning is generally transparent to SQL, but partitioning has also some indirect impact on SQL due to a restriction that was placed on the UPDATE statements. The restriction was that

the columns of a partitioning index could *not* be updated. The option to remove this restriction was introduced in DB2 Version 6 with the DSNZPARM field PARTKEYU in panel DSNTIPB.

PARTKEYU can have the following values:

- ▶ NO: This eliminates the ability to update partitioning key columns.
- ▶ SAME: This allows the updating of key columns, but only within the same partition.
- ▶ YES: This allows full update activity against the key columns without restrictions, and it is the default.

Any attempt to update a key that violates the PARTKEYU DSNZPARM results in an SQLCODE -904 error message.

Be aware that using YES or SAME will allow users to update primary keys that could invalidate logical design rules.

When planning to partition, it is essential that the application code be fully understood. If there is updating of the “soon-to-be” partitioning key, then these programs will have to be changed, or the PARTKEYU DSNZPARM field must be set to YES. If the change is made without application checking, applications that worked before will stop working.

If the application cannot be changed, and the value of PARTKEYU remains NONE, then an artificial key must be chosen to act as the partitioning key and be built into the design of the database.





## Simplifying utilities with wildcarding and templates

DB2 provides support for the use of pattern matching and dynamic allocation of data sets required by utilities. This alleviates the need for multiple invocations of utilities for similarly named objects and their removal from the JCL of utility data sets. This combination improves DBA productivity by largely reducing tasks such as JCL writing.

In this chapter, we describe the concepts of and discuss the new template switching function with specific implementation details.

The chapter contains:

- ▶ Wildcards
- ▶ Templates
- ▶ Combining wildcards and templates
- ▶ OPTIONS
- ▶ Using Unicode control statements

## 3.1 Wildcards

This section shows how to use the LISTDEF statements to provide wildcarding, and the benefits of their use.

### 3.1.1 What is wildcarding

Wildcarding provides the ability to define a list of DB2 objects, based upon a pattern rather than explicit names, and to assign a name to that list, via the LISTDEF statement. The list name makes the list available for subsequent execution as the object of a utility control statement or as an element of another LISTDEF.

DB2 automatically generates a list of objects that matches the supplied pattern, and passes the list to one or more specific utilities for processing.

### 3.1.2 Benefits of using wildcards

These are the benefits that the LISTDEF control statement provides:

- ▶ The LISTDEF statement can contain a list of specific objects, a generic expression, or both.
- ▶ Objects in the list can be either included or excluded to match requirements.
- ▶ Housekeeping is isolated from the effects of the introduction of new objects and dropped objects in the DB2 catalog.
- ▶ Consistency points across multiple objects can be ensured by including all logically related objects in a single LISTDEF.
- ▶ Referential integrity (RI) sets can be processed in a single statement.

### 3.1.3 How to use wildcarding

Wildcarding is activated by the inclusion of the LISTDEF statement in the utility job stream, prior to the utility control cards. The LISTDEF can be specified either in the SYSIN ddname or in a library allocated to the SYSLISTD ddname. The default *ddname* can be overridden via the OPTIONS statement.

A *list-name* is allocated to the LISTDEF as part of the control statement. If both the SYSIN and SYSLISTD are included, and the same list-name is found in both, then the instream LISTDEF is used.

Objects can be specified within the LISTDEF at database through to index level. Which objects are returned depends upon whether table spaces or indexes have been requested. The various results are shown in Table 3-1.

Table 3-1 Object results table by keyword

Object type	TABLESPACE keyword results	INDEXSPACE keyword results
Database	All table spaces within a database	All index spaces within a database
Table space	Specified table space	All index spaces related to tables in that table space
Table	Table space containing the table	All index spaces related to the table
Index space	Table space containing a related table	Specified index space

Object type	TABSPACE keyword results	INDEXSPACE keyword results
Index	Table space containing a related table	Index space containing the index
List of table spaces	Table space from the expanded referenced list	Related index spaces for the table spaces in the expanded referenced list
LIST of index spaces	Related table spaces for the index spaces in the expanded referenced list	index spaces from the expanded referenced list
List of table space and index space	Table spaces from the expanded referenced list and related table spaces for the index spaces in the same list	Index spaces from the expanded referenced list and the related index spaces for the table spaces in the same list

Objects are added to the list via the keyword INCLUDE and removed from the built list via EXCLUDE. If an EXCLUDE attempts to remove an object that has not been included, then the EXCLUDE is ignored.

Lists generated by the LISTDEF statement are ordered, but are not sorted, and do not contain duplicates. If restart is required, then a checkpoint list is used at restart.

**Important:** A secondary INCLUDE statement may return a previously EXCLUDED object to the list.

Here are examples comparing the use of LISTDEF to the method where you do not use them. Example 3-1 will have to be altered for every new object that is required to be quiesced in database DBA.

*Example 3-1 Not using LISTDEF*

---

```
//SYSIN DD *
QUIESCE TABSPACE DBA.X
        TABSPACE DBA.Y
        TABSPACE DBA.Z
```

---

No changes are necessary on DB2 9 (see Example 3-2).

*Example 3-2 Version 9*

---

```
//SYSIN DD *
LISTDEF DBA INCLUDE TABSPACE DBA.*
QUIESCE LIST DBA
```

---

As shown in Example 3-3, there is also the ability to EXCLUDE objects from the list of objects passed on to the utility.

*Example 3-3 Excluding objects*

---

```
//SYSIN DD *
LISTDEF DBA INCLUDE TABSPACE DBA.*
        EXCLUDE TABSPACE DBA.Z
QUIESCE LIST DBA
```

---

The LIST keyword is supported by the utilities listed in Table 3-2, and where possible, the utility optimizes the list processing.

*Table 3-2 Pattern-matching character comparison table*

Utility	Method of processing
CHECK INDEX	Grouped by related table space
COPY	Processed in order specified on a single call to COPY
COPY PARALLEL	Keyword is supported, objects in the list are sorted by descending size
COPYTOCOPY	Processed in order specified on a single call to COPYTOCOPY
MERGECOPY	Processed in order specified
MODIFY RECOVERY	Processed in order specified
MODIFY STATISTICS	Processed in order specified
QUIESCE	Processed in order specified on a single call to QUIESCE
REBUILD	Grouped by related table space
RECOVER	Processed in order specified on a single call to RECOVER
RECOVER PARALLEL	Keyword is supported, objects in the list are sorted by descending size
REORG	Processed in order specified
REPORT	Processed in order specified
RUNSTATS INDEX	Grouped by related table space
RUNSTATS TABLESPACE	Processed in order specified
UNLOAD	Grouped by table space at the partition level

Further examples are displayed under the utilities that are able to accept LISTDEF statements. For a full list of utilities that can use LISTDEF, refer to Figure 3-6 on page 68.

### 3.1.4 Using multiple wildcards

Another feature of LISTDEF is the ability to specify more than one set of wildcards in an input stream, whether instream or via SYSLISTD. Each LISTDEF is processed and a list is generated for each one, but any list generated is only processed when it is referenced by a utility statement.

Be aware that each LISTDEF containing a wildcard results in calls to the DB2 Catalog to satisfy the list. Thus, jobs may run longer than necessary when a large number of LISTDEFs are present in one job and are not referenced. Therefore, while including all LISTDEFs in every utility job would appear simpler, care should be taken with this approach.

A LISTDEF statement is only expanded once; therefore, it can be referenced by many utilities in the job step without incurring the cost of expansion multiple times.



### 3.1.5 LISTDEF expansion

When DB2 expands your list definition, that is, when it generates a list of objects according to your LISTDEF statement, the sequence of steps DB2 uses influences the result. You must know this sequence in order to code more complicated list definitions correctly.

Here is a simple example: five table spaces and their unique indexes reside in the two databases DBX and DBY. The partitioned table space PTS1 has an additional non-partitioned index IXNPP1. After the previous recovery of a table space set to a prior point in time, because the indexes have not been recovered to the same point in time, a rebuild of all indexes on all tables in all table spaces of this table space set must be completed. Normally, if no COPY enabled indexes are involved, the first INCLUDE clause of the LISTDEF statement shown in Figure 3-1 contains the statements needed to fulfill this task.

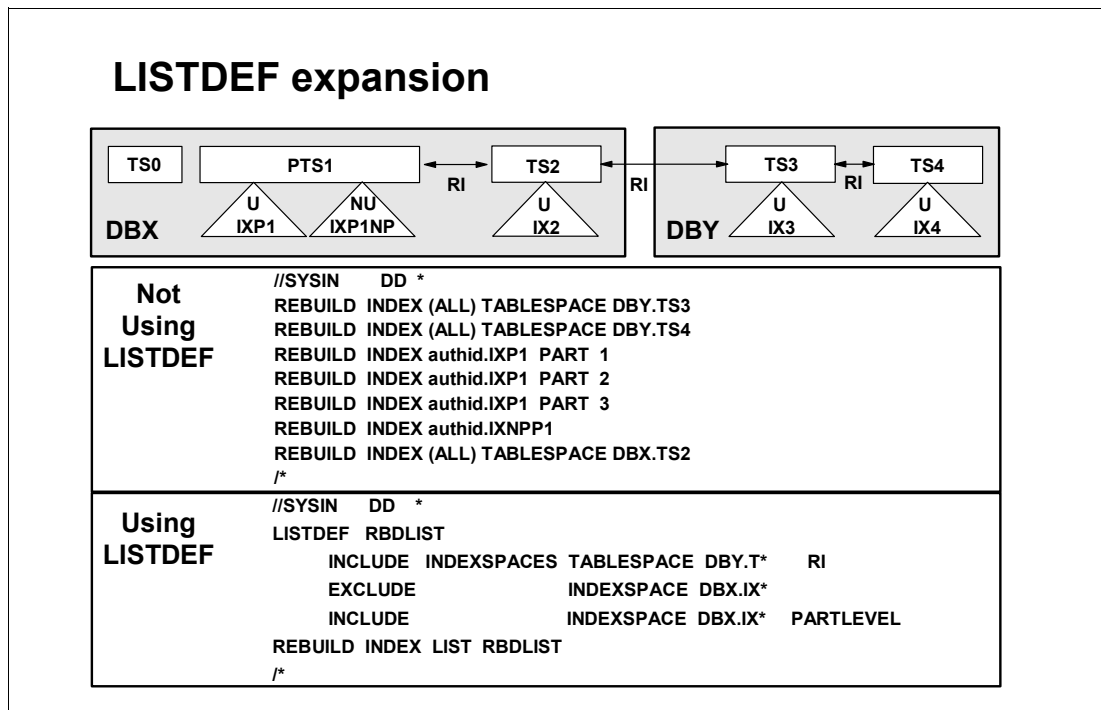


Figure 3-1 LISTDEF scenario

An example of the complete recovery job with LISTDEF is listed in Example 3-4.

Example 3-4 Sample LISTDEF: recovery job

```
LISTDEF RECLIST INCLUDE TABLESPACE DBY.TS% RI
LISTDEF RBDLIST INCLUDE INDEXSPACES TABLESPACE DBY.T% RI
RECOVER LIST RECLIST TOLOGPOINT X'xxxxxxxxxxxxxxxxxx'
REBUILD INDEX LIST
RBDLIST
```

A further example is one where we assume that all partitioned indexes are to be rebuilt, per partition, and that they reside in database DBY. If all partitioned index space names in DBX start with 'IXP', then Figure 3-1 shows a way that possible with DB2 9. Again, the advantage is that you do not have to change the DB2 9 job if table spaces or indexes are dropped or added, as long as established naming conventions are followed.

The LISTDEF consists of four steps, as shown in Figure 3-2.

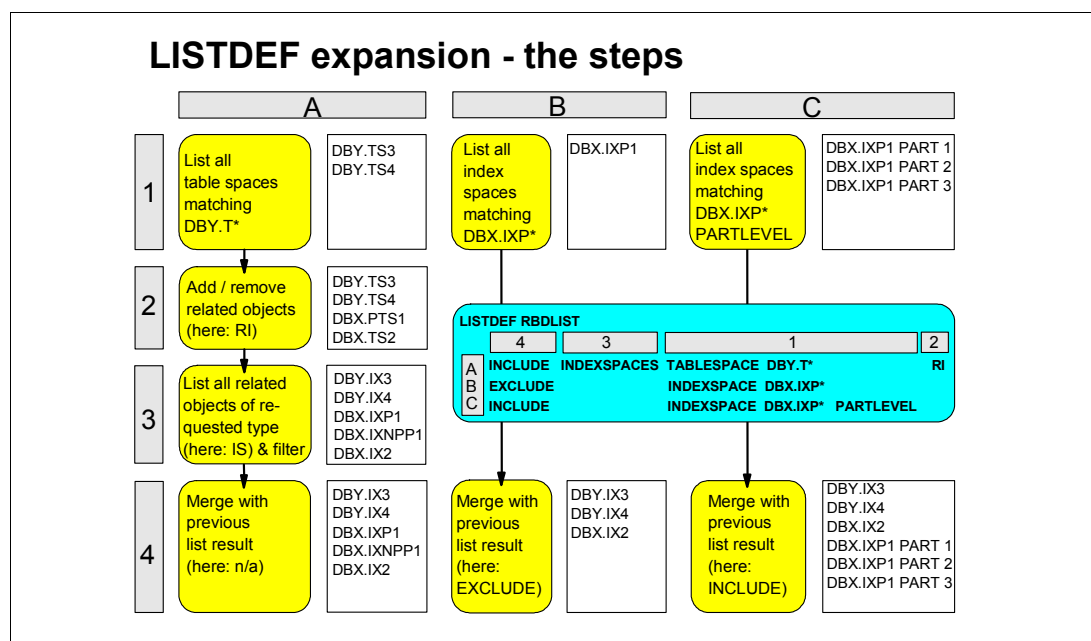


Figure 3-2 LISTDEF expansion

Where:

1. An initial catalog lookup is performed to find and list the explicitly specified object or objects that match the specified pattern. In our example, in the first INCLUDE clause, table spaces are searched that match the pattern DBY.T\*. The two matching table spaces, DBY.TS3 and DBY.TS4, are listed in the example at the right side of step 1.
2. Related objects are added or removed depending on the presence of the RI, BASE, LOB, XML, or ALL keywords. Two types of relationships are supported: referential and auxiliary relationships. More specifically:
  - If you specify RI, then the TABLESPACESET process is invoked and referentially related objects are added to the list. As a result, all referentially connected table spaces, by their tables, are included in the list. Figure 3-2 shows these four table spaces.
  - If you specify LOB, XML or ALL, then auxiliary related objects are added to the list.
  - If you specify BASE or ALL, then auxiliary related base objects are added to the list.
  - If you specify LOB or XML, then base objects are excluded from the list.
  - If you specify BASE, then auxiliary objects are excluded from the list.
3. This step consists of two substeps:
  - a. All related objects of the requested type are searched for and added to the list. This step can be skipped if the list built so far already consists of objects of the requested type, either table spaces or index spaces. Otherwise, the two cases must distinguished:
    - If a list of table spaces is required, that is, the type specification is TABLESPACES, but the initial catalog lookup has been done with another type of object, for example, with index spaces or indexes, then related table spaces are added to the list. Obviously, those table spaces are related, where the base tables reside in the indexes or index spaces already in the list.

- If a list of index spaces is required, that is, the type specification is INDEXSPACES, but the initial catalog lookup has been done with another type of object, for example, with table spaces or tables, then the related index spaces are added to the list.
  - In Figure 3-2 on page 52, in the first INCLUDE clause, the four table spaces have five index spaces all together. These index spaces are added to the list.
- b. A filtering process takes place:
- If INCLUDE TABLESPACES is specified, then all objects that are not table spaces are removed from the list. Analog rules apply for INCLUDE INDEXSPACES, EXCLUDE TABLESPACES, and EXCLUDE INDEXSPACES.
  - The specification of COPY YES or NO is also taken into account in this step.
  - In Figure 3-2 on page 52, in the first INCLUDE clause, INCLUDE INDEXSPACES, has been specified. Therefore, all table spaces are removed from the list. The list now contains five index spaces only, as you can see in the diagram.
4. If the keyword INCLUDE is specified, the objects of the resulting list are added to the list derived from the previous INCLUDE or EXCLUDE clauses, if they are not in the list already. In other words, an INCLUDE of an object already in the list is ignored; the list contains no duplicates. If the keyword EXCLUDE is specified, the objects of the resulting list are removed from the list derived from the previous INCLUDE or EXCLUDE clause. An EXCLUDE of an object not in the list is ignored. In Figure 3-2 on page 52, for the first INCLUDE clause (A), there is no previous list to merge with, so this step is skipped.

All four steps are explained for the first sample INCLUDE clause (A). Next, the EXCLUDE clause (B) and then the second INCLUDE clause (C) are processed. This is processed in a similar way as described above:

- Step 1 is similar to step 1 (already explained) of the first INCLUDE clause (A).
- Step 2 is skipped, as no relationship type was specified, neither RI, ALL, BASE, or LOB.
- Step 3 is skipped, as spec-type INDEXSPACES is assumed if obj-type is INDEXSPACE. Therefore, there is no need to look for related objects or to filter out objects of another type.
- In step 4 for the EXCLUDE clause (B), two objects are removed from the list generated by the first INCLUDE clause (A).
- In step 4 for the second INCLUDE clause (C), four objects are added to the list generated after the EXCLUDE clause (B).

**Note:** The purpose of the last example is to conceptually present the resolution of a LISTDEF statement in order to help with defining LISTDEF statements properly. The purpose is not to precisely document DB2's implementation of the sequence of the expansion steps (which is in fact different), because we want this process to be generally applicable, and not specific for the example. DB2 may perform step 2 after step 3, for example, if you specify INCLUDE TABLESPACES INDEX authid.ix RI. Furthermore, the PARTLEVEL keyword may be postponed to the end, that is, after step 3.

A sample of the utility output is shown in Example 3-5, which shows the expansion of the LISTDEF.

*Example 3-5 Sample output*

---

```
- OUTPUT START FOR UTILITY, UTILID = TEMP
- OPTIONS PREVIEW
- PROCESSING CONTROL STATEMENTS IN PREVIEW MODE
- OPTIONS STATEMENT PROCESSED SUCCESSFULLY
- LISTDEF RBDLIST INCLUDE INDEXSPACES TABLESPACE DBY.T* RI
  EXCLUDE INDEXSPACE DBX.IXP* INCLUDE INDEXSPACE DBX.IXP* PARTLEVEL
- LISTDEF STATEMENT PROCESSED SUCCESSFULLY
  - EXPANDING LISTDEF RBDLIST
    - PROCESSING INCLUDE CLAUSE TABLESPACE DBY.T*    <- A: INCLUDE
    - CLAUSE IDENTIFIES 5 OBJECTS <- after step 3
    - PROCESSING EXCLUDE CLAUSE INDEXSPACE DBX.IXP* <- B: EXCLUDE
    - CLAUSE IDENTIFIES 2 OBJECTS <- after step 3
    - PROCESSING INCLUDE CLAUSE INDEXSPACE DBX.IXP*    <- C: INCLUDE
    - CLAUSE IDENTIFIES 4 OBJECTS <- after step 3
    - LISTDEF RBDLIST CONTAINS 7 OBJECTS <- final list
- LISTDEF RBDLIST EXPANDS TO THE FOLLOWING OBJECTS:
LISTDEF RBDLIST -- 00000007 OBJECTS
  INCLUDE INDEXSPACE DBY.IX3
  INCLUDE INDEXSPACE DBY.IX4
  INCLUDE INDEXSPACE DBX.IX2
  INCLUDE INDEXSPACE DBX.IXP1 PARTLEVEL(00001)
  INCLUDE INDEXSPACE DBX.IXP1 PARTLEVEL(00002)
  INCLUDE INDEXSPACE DBX.IXP1 PARTLEVEL(00003)
DSNUGUTC - REBUILD INDEX LIST RBDLIST
DSNUGULM - PROCESSING LIST ITEM: INDEXSPACE DBY.IX3
DSNUGULM - PROCESSING LIST ITEM: INDEXSPACE DBY.IX4
DSNUGULM - PROCESSING LIST ITEM: INDEXSPACE DBX.IXP1
DSNUGULM - PROCESSING LIST ITEM: INDEXSPACE DBX.IXP1
DSNUGULM - PROCESSING LIST ITEM: INDEXSPACE DBX.IXP1
DSNUGULM - PROCESSING LIST ITEM: INDEXSPACE DBX.IX2
DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

---

### 3.1.6 Recommendations for the use of wildcards

Wildcarding should be used wherever possible to isolate the JCL from changes to DB2 objects. This directly results in less effort being required to ensure the integrity of the DB2 instance. You should:

- ▶ Group logically related objects into the same LISTDEF to ensure consistency.
- ▶ Use partitioned data set (PDS) members as input to reduce maintenance across JCL and to ensure consistency.
- ▶ Ensure that all databases are referenced by at least one LISTDEF, or are explicitly coded.
- ▶ Revisit LISTDEFs periodically to ensure the integrity and completeness of the definitions. Naming standards for user defined objects would assist in this process and allow for even less maintenance of LISTDEF.
- ▶ Use OPTION PREVIEW mode to review the output from LISTDEF.
- ▶ Use in conjunction with templates. See 3.2, “Templates” on page 55 for the full benefits.

### 3.1.7 Current restrictions in the use of wildcards

The use of wildcarding cannot be used against the catalog or directory objects. If either of these are required in a LISTDEF statement, then these have to be explicitly coded.

The options RI and PARTLEVEL are mutually exclusive.

Unless the OPTION statement is used, if the execution of the utility fails due to a restriction on an object, then all objects in the list are not processed. By using OPTION EVENT(ITEMERROR, SKIP), this situation can be avoided.

LISTDEF is not supported by the following utilities:

- ▶ CATMAINT
- ▶ CHECK DATA
- ▶ DIAGNOSE
- ▶ LOAD
- ▶ REPAIR
- ▶ STOSPACE

## 3.2 Templates

Templates are a method of defining data set masks that are to be used by utilities to dynamically allocate data sets required for successful execution of the utility. They can be used for work data sets as well as utility data sets, for example, image copy data sets. When used in conjunction with LISTDEF, this provides a powerful mechanism for executing utilities, allows faster development of job streams, and requires fewer modifications when the underlying list of database objects changes.

Templates allow the writing of data sets to both disk and tape units and incorporate all the features normally coded within the DD statement. This also allows for the stacking of data sets on tapes automatically. The TEMPLATE statement executes entirely in the UTILINIT phase in preparation for the subsequent utility. Output from the TEMPLATE control statement is a dynamic allocation TEMPLATE with an assigned name for later reference.

The use of templates also introduces the concept of automatic intelligent data set sizing. For this to be efficient, RUNSTATS have to be current.

### 3.2.1 Benefits of using templates

The benefits of using templates include:

- ▶ Less reliance on specialist JCL knowledge
- ▶ Frees DBA resource for more important tasks
- ▶ Enforcement of naming standards
- ▶ Standardization of data set allocations
- ▶ Multiple TEMPLATES per job
- ▶ Used for workfiles
- ▶ Changes to naming standards quickly and easily implemented
- ▶ Automatic data set sizing
- ▶ Used alongside LISTDEF for full flexibility
- ▶ Automatically defines GDG bases where required

## 3.2.2 How to use templates

Templates can be used to eliminate the necessity of needing DD statements within the JCL, allowing for standardized JCL to be used for all the utilities that support the use of templates.

Templates are initiated by using the **TEMPLATE** keyword in the utility job stream before the utility commands. Like **LISTDEF**, the **TEMPLATE** can either be coded within the **SYSIN** ddname, or in a data set allocated to the job stream via the default **SYSTEMPL** ddname. This is the default and can be overridden by using the **OPTIONS** keyword.

The minimum requirement for a **TEMPLATE** statement is a name, which is referenced by the utility command, and the data set naming mask. If no sizing information is supplied, then DB2 calculates the space required for the data set. See Example 3-6.

*Example 3-6 Template not invoking data set sizing*

---

```
TEMPLATE LOCALDDN
  DSN(DB2V910G.&DB..&TS..D&DATE..T&TIME.L)
```

---

If the **TEMPLATE** supplies any other parameters, then these parameters take precedence. See Example 3-7.

*Example 3-7 Template invoking data set sizing*

---

```
TEMPLATE LOCALDDN
  DSN(DB2V910G.&DB..&TS..D&DATE..T&TIME.L)
  UNIT(SYSDA) SPACE(15,15) TRK DISP(NEW,CATLG,CATLG)
  VOLUMES(SBOX57)
```

---

Templates can be used to write to either disk or tape. Support is included for stacking multiple files onto a single tape via the **STACK** option; see Example 3-8. Support for DF/SMS is also included.

*Example 3-8 Template using tape with STACK option*

---

```
TEMPLATE REMOTDDN
  DSN(DB2V910G.&DB..&TS..D&DATE..T&TIME.R)
  UNIT ATL2 VOLUMES(TST002) STACK YES
```

---

There can be more than one **TEMPLATE** per job stream (see Example 3-9). Each one must have a unique name, and a utility can reference more than one **TEMPLATE** if that utility supports multiple output ddnames, for example, **COPY**.

*Example 3-9 Using multiple TEMPLATES*

---

```
TEMPLATE LOCALDDN
  DSN(DB2V910G.&DB..&TS..D&DATE..T&TIME.L)
  UNIT(SYSDA) SPACE(15,15) TRK DISP(NEW,CATLG,CATLG)
  VOLUMES(SBOX57)
TEMPLATE REMOTDDN
  DSN(DB2V910G.&DB..&TS..D&DATE..T&TIME.R)
  UNIT ATL2 VOLUMES(TST002) STACK YES
LISTDEF DB01A
  INCLUDE TABLESPACE DBX.TSY
COPY LIST DB01A FULL YES
  SHRLEVEL CHANGE
  COPYDDN(LOCALDDN) RECOVERYDDN(REMOTDDN)
```

---

If the DISPOSITION option is not specified in the TEMPLATE definition, DB2 uses the disposition defaults for the utility concerned. DB2 also takes into account any restart implications. For an example of using REORG, see Figure 3-3.

Example: REORG

		status	normal termination	abnormal termination
New utility	SYSREC	NEW	CATLG	CATLG
execution	SYSUT1	NEW	DELETE	CATLG
	SORTOUT	NEW	DELETE	CATLG
Restarted	SYSREC	MOD	CATLG	CATLG
utility	SYSUT1	MOD	DELETE	CATLG
	SORTOUT	MOD	DELETE	CATLG

Figure 3-3 Example disposition with TEMPLATES

Further examples of templates are given under the relevant utility sections.

**Note:** The DISP field in TEMPLATES does *not* support the typical default MVS syntax, such as DISP=(,KEEP). In TEMPLATES, *all* three parameters must be specified.

### 3.2.3 Naming standards

The TEMPLATE control statement defines the data set naming convention and it can optionally contain allocation parameters that define data set size, location, and attributes. Variables are supported that allow data sets to be built dynamically. DB2 does not check to ensure that they are unique until the utility executes. Therefore, you must be careful when constructing data set names to ensure uniqueness. Normal z/OS data set naming conventions apply. More specifically, all qualifiers must begin with an alphabetical character and the length cannot exceed 44 characters. The use of templates allows for the standardization of data set names across the DB2 subsystem, and allows easy identification of the data set type created, if the relevant variables are used in construction of the name, for example, &IC. for image copy.

A full list of the currently allowable variables that can be used within the TEMPLATE statement is provided in Table 3-3.

Table 3-3 *TEMPLATE variables*

<b>JOB variables</b>	<b>UTILITY variables</b>	<b>OBJECT variables</b>	<b>DATE and TIME variables</b>
JOBNAME	UTIL (utility name truncated)	DB (dbname)	DATE (yyyyddd)
STEPNAME	ICTYPE ("F" or "I")	TS (table space name)	YEAR (yyyy)
USERID or US	LOCREM or LR ("L" or "R")	IS (index space name)	MONTH (mm)
SSID (subsystem ID or DS group attach name)	PRIBAC or PB ("P" or "B")	SN (space name)	DAY (dd)
		PART (five digits)	JDATE or JU (Julian date yyyyddd)
		LIST (name of the list)	JDAY (ddd)
		SEQ (sequence number of the object in the list)	TIME (hhmmss)
			HOUR
			MINUTE
			SECOND or SC

### 3.2.4 Substring notation support

In order to help users with their own naming standards and the length limitation, you can use of substring notation for the data set name. If you use this notation, the entire DSN operand must be enclosed in single quotes.

To specify a substring, use the form `&variable(start)`. or `&variable(start,length)`, where:

► **start**

Specifies the substring's starting byte location within the current variable base value at the time of execution. start must be an integer from 1 to 128.

► **length**

Specifies the length of the substring.

If you specify start, but do not specify length, length defaults to the number of characters from the character specified by start to the last character of the variable value at the time of execution. length must be an integer that does not cause the substring to extend beyond the end of the base value.



Examples:

- ▶ &SSID(2,3). would return the variable &SSID. value starting with character 2 for a length of 3 characters.
- ▶ &PART(2). would return the variable &PART. value starting with character 2 for the remainder of the variable value.

A common use of substring is for &PART., because five digits are seldom necessary. Also notice that it is not possible to substring a member name because variables inside the parentheses, substringed or not, are not supported.

A simple example of an Image Copy job using substring notation is shown in Example 3-10.

---

*Example 3-10 Template with substring notation*

---

```
//COPY1 EXEC DSNUPROC,SYSTEM=D9C3,UID=' '  
//DSNUPROC.SYSIN DD *  
TEMPLATE COPYDDN1 UNIT(SYSDA)  
DSN('S&DB(3,4)..IC12.&DB..&TS.')
```

---

```
DISP(MOD,KEEP,KEEP)  
TEMPLATE COPYDDN2 UNIT(SYSDA)  
DSN('S&DB(3,4)..IC22.&DB..&TS.')
```

---

```
DISP(MOD,KEEP,KEEP)  
LISTDEF J0000004  
INCLUDE TABLESPACE DSN8D91A.DSN8S91E  
INCLUDE TABLESPACE RGDB02.GLWSEMP  
COPY LIST J0000004 COPYDDN(COPYDDN1,COPYDDN2)
```

---

The generated data set name is shown in the output listed in Example 3-11.

---

*Example 3-11 Output job with substring*

---

```
14:23:08.16 DSNUILDR - LISTDEF STATEMENT PROCESSED SUCCESSFULLY  
14:23:08.16 DSNUGUTC - COPY LIST J0000004 COPYDDN(COPYDDN1, COPYDDN2)  
14:23:08.20 DSNUGULM - PROCESSING LIST ITEM: TABLESPACE DSN8D91A.DSN8S91E  
14:23:08.20 DSNUGULM - PROCESSING LIST ITEM: TABLESPACE RGDB02.GLWSEMP  
14:23:08.82 DSNUGDYN - DATASET ALLOCATED. TEMPLATE=COPYDDN1  
DDNAME=SYS00001  
DSN=SN8D9. IC12. DSN8D91A. DSN8S91E  
14:23:08.91 DSNUGDYN - DATASET ALLOCATED. TEMPLATE=COPYDDN2  
DDNAME=SYS00002  
DSN=SN8D9. IC22. DSN8D91A. DSN8S91E  
14:23:09.01 DSNUBBID - COPY PROCESSED FOR TABLESPACE DSN8D91A.DSN8S91E  
NUMBER OF PAGES=16  
AVERAGE PERCENT FREE SPACE PER PAGE = 8.87  
PERCENT OF CHANGED PAGES = 12.50  
ELAPSED TIME=00:00:00
```

---

For details, see the *DB2 Version 9.1 for z/OS Utility Guide and Reference*, SC18-9855.

### 3.2.5 Intelligent data set sizing

To use DB2 automatic data set sizing, templates *must* be used. As stated earlier, if the space parameters are missing from the TEMPLATE statement, then DB2 estimates the sizes of the data sets based on a utility-specific formula. For this feature to work successfully, the object statistics have to be kept up-to-date.

The input values used in these formulas mostly come from the DB2 catalog tables. The high-used RBA is read at open time and it is maintained by the buffer manager. It is the most current value, updated before it is even written to the ICF catalog.

All these formulas are documented in the standard DB2 manuals. Figure 3-4 is an example of the specific formulas for the data sets for the REORG utility, in cases where you use:

- ▶ REORG UNLOAD CONTINUE SORTDATA KEEPDICTIONARY SHRLEVEL NONE
- ▶ SHRLEVEL REFERENCE

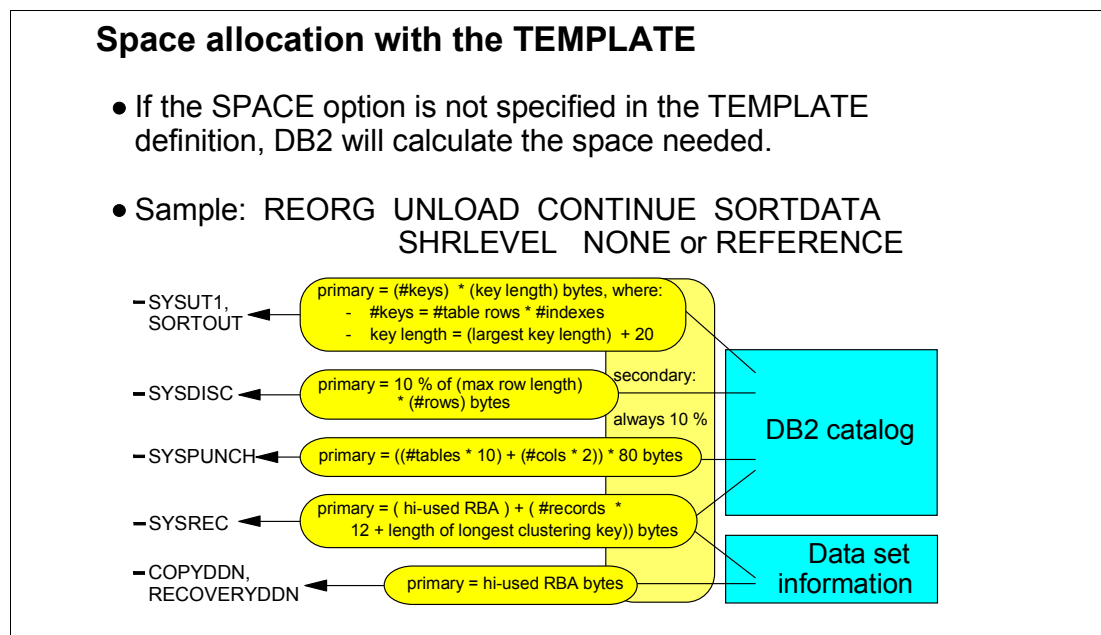


Figure 3-4 Automatic data set sizing for the REORG utility

### 3.2.6 Template switching function

This new function allows image copies of varying sizes to have different characteristics. Those image copies should be produced by:

- ▶ COPY
  - FULL YES / NO
  - CONCURRENT
- ▶ COPYTOCOPY
- ▶ MERGECOPY
- ▶ LOAD
- ▶ REORG

This provides significant flexibility in terms of data set names and attributes, for example, for device types.

The LIMIT keyword controls this function. There are two mandatory operands:

- ▶ The maximum primary allocation that is permitted for the template
- ▶ The new template to be used when this maximum is reached

Template switching can only be done once per data set allocation.

In Example 3-12, we show how to use keyword LIMIT in TEMPLATE

*Example 3-12 Use of keyword LIMIT*

---

```
//SYSIN DD *
TEMPLATE LESS_80 DSN &DB..&TS..IC..D&DA..T&TI. UNIT(SYSDA) LIMIT(80 CYL,GREAT_80)
TEMPLATE GREAT_80 DSN &DB..&TS..IC..D&DA..T&TI. UNIT=TAPE
COPY TABLESPACE DSN8S91E.DSN8D91A COPYDDN(LESS_80)
COPY TABLESPACE AUXD501.AUXSITH COPYDDN(LESS_80)
```

---

In Example 3-13, if table space DSN8S91E is less than 80 cyl, then it can be accommodated within the LESS\_80 TEMPLATE, but the table space AUXSITH is bigger than 80 cyl and was switched to the GREAT\_80 TEMPLATE. The image copy data set is then allocated on tape.

*Example 3-13 LIMIT keyword output*

---

```
17.05.58 JOB17263 ---- FRIDAY, 25 SEP 2009 ----
17.05.58 JOB17263 IRR010I USERID DB2R7 IS ASSIGNED TO THIS JOB.
17.05.58 JOB17263 ICH70001I DB2R7 LAST ACCESS AT 16:33:55 ON FRIDAY, SEPTEM
17.05.58 JOB17263 $HASP373 DB2R7001 STARTED - INIT 1 - CLASS A - SYS SC63
17.05.58 JOB17263 IEF403I DB2R7001 - STARTED - TIME=17.05.58 - ASID=0020 - SC6
17.05.58 JOB17263 *IEF233D M OB03,PRIVAT,SL,DB2R7001,DSNUPROC, 385
385 AUXD501.AUXSITH.FD25.T210621,
385 OR RESPOND TO IEF455D MESSAGE
17.05.58 JOB17263 *062 IEF455D MOUNT PRIVAT ON OB03 FOR DB2R7001 DSNUPROC OR RE
17.07.28 JOB17263 IEC705I TAPE ON OB03,TAPE01,SL,COMP,DB2R7001,DSNUPROC.COPY1,
17.07.47 JOB17263 IEC205I SYS00002,DB2R7001,DSNUPROC,FILESEQ=1, COMPLETE VOLUM
394 DSN=AUXD501.AUXSITH.FD25.T210621,VOLS=TAPE01,TOTALBLOCKS=200
17.07.47 JOB17263 IEF234E K OB03,TAPE01,PVT,DB2R7001,DSNUPROC
17.07.47 JOB17263 - --TIMINGS (MINS.)-
17.07.47 JOB17263 -JOBNAME STEPNAME PROCSTEP RC EXCP CPU SRB CLOC
17.07.47 JOB17263 -DB2R7001 COPY1 DSNUPROC 00 20776 .02 .00 1.8
17.07.47 JOB17263 IEF404I DB2R7001 - ENDED - TIME=17.07.47 - ASID=0020 - SC63
```

---

And the output is :

```
17:05:58.23 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = DB2R7.DB2R7001
17:05:58.28 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
17:05:58.28 DSNUGUTC - TEMPLATE LESS_80 DSN &DB..&TS..&IC.D&DA..T&TI. UNIT=SYSD

17:05:58.28 DSNUJTDR - TEMPLATE STATEMENT PROCESSED SUCCESSFULLY
17:05:58.28 DSNUGUTC - TEMPLATE GREAT_80 DSN &DB..&TS..&IC.D&DA..T&TI. UNIT=B03
17:05:58.28 DSNUJTDR - TEMPLATE STATEMENT PROCESSED SUCCESSFULLY
17:05:58.28 DSNUGUTC - COPY TABLESPACE DSN8D91A.DSN8S91E COPYDDN(LESS_80)
17:05:58.32 DSNUGDYN - DATASET ALLOCATED. TEMPLATE=LESS_80
DDNAME=SYS00001
DSN=DSN8D91A.DSN8S91E.FD25.T210621
17:05:58.39 DSNUBBID - COPY PROCESSED FOR TABLESPACE DSN8D91A.DSN8S91E
NUMBER OF PAGES=16
AVERAGE PERCENT FREE SPACE PER PAGE = 8.75
```

```

        PERCENT OF CHANGED PAGES = 0.00
        ELAPSED TIME=00:00:00
268 17:05:58.40 DSNUBAFI - DB2 IMAGE COPY SUCCESSFUL FOR TABLESPACE DSN8D91A.DSN
17:05:58.41 DSNUGUTC - COPY TABLESPACE AUXD501.AUXSITH COPYDDN(LESS_80)

17:07:24.26 DSNUGDYN - DATASET ALLOCATED. TEMPLATE=GREAT_80
        DDNAME=SYS00002, FILE SEQUENCE=0001
        DSN=AUXD501.AUXSITH.FD25.T210621
17:07:47.77 DSNUBBID - COPY PROCESSED FOR TABLESPACE AUXD501.AUXSITH
        NUMBER OF PAGES=140389
        AVERAGE PERCENT FREE SPACE PER PAGE = 28.23
        PERCENT OF CHANGED PAGES = 0.00
        ELAPSED TIME=00:01:49
268 17:07:47.78 DSNUBAFI - DB2 IMAGE COPY SUCCESSFUL FOR TABLESPACE
AUXD501.AUXSITH
17:07:47.79 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0

```

---

**Tip:** Do not forget that the **TEMPLATE** name can have up to eight alphanumeric characters and must begin with an alphabetic character.

### 3.2.7 Template with LOBs and file reference variables

File reference variables were introduced for LOB usage with DB2 9. We illustrate the use of templates with file reference variables and the UNLOAD utility. For more information about and examples of LOBs, refer to *LOBs with DB2 for z/OS: Stronger and Faster*, SG24-7270.

For unload, DB2 creates or uses a different output file for each LOB value to be unloaded. The output file can be one of the following types:

- ▶ Member of a partitioned data set (PDS) or partitioned data set extended (PDSE)
- ▶ Hierarchical File System (HFS) file on a HFS directory

The LOB unload file contains the entire LOB value and the name of this file is stored in the normal unload output file as a CHAR or VARCHAR field.

Additional keywords have been added to the CHAR and VARCHAR field-specifications of the UNLOAD utility to support a file name as an output file to store the actual LOB value:

- ▶ BLOBF: The output field contains the name of a file to store a BLOB field.
- ▶ CLOBF: The output field contains the name of a file to store a CLOB field.
- ▶ DBCLOBF: The output field contains the name of a file to store a DBCLOB field.

For UNLOAD, the actual files to be created or used for storing the LOB values are generated from a **TEMPLATE** definition with a name to be specified along with the BLOBF, CLOBF, or DBCLOBF keywords. The **TEMPLATE** definition is used to specify the characteristics of the LOB unload files.

For a PDS:

- ▶ The name of the PDS is generated from the DSN specification of the template.
- ▶ Specify DSNTYPE PDS.
- ▶ Specify DIR to specify the number of directory blocks for a new PDS.
- ▶ The member names are automatically generated.

For a PDSE:

- ▶ The name of the PDSE is generated from the DSN specification of the template.
- ▶ Specify DSNTYPE LIBRARY.
- ▶ The member names are automatically generated.

For a HFS file:

- ▶ The name of the HFS directory is generated from the DSN specification of the template.
- ▶ Specify DSNTYPE HFS.
- ▶ The HFS directory must exist and be mounted.
- ▶ The data set names in the directory are automatically generated.

For a PDS, the default value of DIR is the estimated number of records divided by 20; it is sufficient to have as many members as the estimated number of records. If you do not specify DSNTYPE, a PDS is created by default (DSORG = PS).

The member names that are generated for a PDS or PDSE or the data set names for a HFS directory have a length of 8 and are uniquely generated from the system clock. They have the same format as when you use the &UNIQ. or &UQ. variables in a TEMPLATE definition.

In Example 3-14, we unload from table ##T.NORMEN00 a whole LOB data of column into a PDS. We specify a TEMPLATE TSYSLOB with DSNTYPE PDS and specify for field IMAGE a field specification as VARCHAR(54) BLOBF TSYSLOB (the maximum possible length for a PDS + membername being 54).

---

*Example 3-14 Unload LOB data to a PDS*

---

```
TEMPLATE TSYSPUN
  DSN('PAOLOR2.&SS.&DB.&SN..UNLOAD.PUNCH6')
  DISP(MOD,CATLG,CATLG)
TEMPLATE TSYSREC
  DSN('PAOLOR2.&SS.&DB.&SN..UNLOAD.SYSRC6')
  DISP(MOD,CATLG,CATLG)
TEMPLATE TSYSLOB
  DSN('PAOLOR2.&SS.&DB.&SN..UNLOAD.PDS6')
  DISP(MOD,CATLG,CATLG)
  DSNTYPE(PDS)
UNLOAD DATA FROM TABLE ##T.NORMEN00
  (DOC_ID,FORMAT,IMAGE VARCHAR(54) BLOBF TSYSLOB)
UNLDDN(TSYSREC) PUNCHDDN(TSYSPUN)
```

---

The results are shown in Example 3-15.

*Example 3-15 Unload LOB data to a PDS*

---

```
DSNU000I 205 18:00:05.58 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = UNLOAD.NORMEN00
DSNU1044I 205 18:00:05.64 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
DSNU050I 205 18:00:05.65 DSNUGUTC - TEMPLATE TSYSFUN DSN('PAOLOR2.&SS.&DB.&SN..UNLOAD.PUNCH6') DISP(MOD,
CATLG, CATLG)
DSNU1035I 205 18:00:05.65 DSNUJTDR - TEMPLATE STATEMENT PROCESSED SUCCESSFULLY
DSNU050I 205 18:00:05.65 DSNUGUTC - TEMPLATE TSYSREC DSN('PAOLOR2.&SS.&DB.&SN..UNLOAD.SYSRC6') DISP(MOD,
CATLG, CATLG)
DSNU1035I 205 18:00:05.65 DSNUJTDR - TEMPLATE STATEMENT PROCESSED SUCCESSFULLY
DSNU050I 205 18:00:05.66 DSNUGUTC - TEMPLATE TSYSLOB DSN('PAOLOR2.&SS.&DB.&SN..UNLOAD.PDS6') DISP(MOD,
CATLG,CATLG) DSNTYPE(PDS)
DSNU1035I 205 18:00:05.66 DSNUJTDR - TEMPLATE STATEMENT PROCESSED SUCCESSFULLY
DSNU050I 205 18:00:05.66 DSNUGUTC - UNLOAD DATA
DSNU650I -DB9B 205 18:00:05.66 DSNUUGMS - FROM TABLE ##T.NORMEN00
DSNU650I -DB9B 205 18:00:05.66 DSNUUGMS - (DOC_ID,
DSNU650I -DB9B 205 18:00:05.66 DSNUUGMS - FORMAT,
DSNU650I -DB9B 205 18:00:05.66 DSNUUGMS - IMAGE VARCHAR(54) BLOBF TSYSLOB) UNLDDN(TSYSREC)
PUNCHDDN(TSYSFUN)
DSNU1038I 205 18:00:05.75 DSNUGDYN - DATASET ALLOCATED. TEMPLATE=TSYSREC
DDNAME=SYS00001
DSN=PAOLOR2.DB9B.NORMEN00.NORMEN00.UNLOAD.SYSRC6
DSNU1038I 205 18:00:05.96 DSNUGDYN - DATASET ALLOCATED. TEMPLATE=TSYSLOB
DDNAME=SYS00002
DSN=PAOLOR2.DB9B.NORMEN00.NORMLOB.UNLOAD.PDS6
DSNU1038I 205 18:02:20.07 DSNUGDYN - DATASET ALLOCATED. TEMPLATE=TSYSFUN
DDNAME=SYS00003
DSN=PAOLOR2.DB9B.NORMEN00.NORMEN00.UNLOAD.PUNCH6
DSNU253I 205 18:02:20.16 DSNUUNLD - UNLOAD PHASE STATISTICS - NUMBER OF RECORDS UNLOADED=5883 FOR TABLE
##T.NORMEN00
DSNU252I 205 18:02:20.16 DSNUUNLD - UNLOAD PHASE STATISTICS - NUMBER OF RECORDS UNLOADED=5883 FOR
TABLESPACE NORMEN00.NORMEN00
DSNU250I 205 18:02:20.16 DSNUUNLD - UNLOAD PHASE COMPLETE, ELAPSED TIME=00:02:14
DSNU568I -DB9B 205 18:02:20.18 DSNUGSRX - INDEX ##T.I_NORMEN00_AUX IS IN INFORMATIONAL COPY PENDING STATE
DSNU010I 205 18:02:20.18 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

---

Three files are created:

- ▶ A punchfile containing the equivalent statements for the LOAD utility
- ▶ A SYSREC file containing the non-LOB data values and the output file names of the LOB values
- ▶ A PDS containing the actual LOB values

### 3.2.8 TEMPLATE with UNIX file (HFS) or pipe

Another enhancement for TEMPLATE is the support of LOAD/UNLOAD from and to a z/OS UNIX file (HFS) or pipe.

APAR PK70269 (UK43948) provides this capacity, but in order to use the TEMPLATE PATH FILEDATA RECORD syntax, the z/OS fixes APARs OA25204, OA25206, and OA25280 must be installed.

If the fix for PK70269 is installed and z/OS does not have the above APARs or the z/OS is lower than release 1.8, the utility fails with return code 8 and an IDCAMS message is issued.

Example 3-16 shows how to use the keyword PATH.

*Example 3-16 Template with UNIX support*

---

```
//DSNUPROC.SYSIN DD *
TEMPLATE DATAFIL RECFM=V LRECL=80
    PATH='/u/marcelo/Loadinput.data'
    PATHOPTS=ORDONLY FILEDATA=RECORD
    PATHDISP=(KEEP,KEEP)
LOAD DATA INDDN DATAFIL LOG NO RESUME YES
    INTO TABLE "db2r7"."marcelo"
```

---

For more information, see 7.16, “Loading and unloading HFS files” on page 200.

The new parameters are:

► FILEDATA

An optional keyword to describe the content type of a z/OS UNIX file. The valid values are:

- TEXT.
- BINARY.
- RECORD. (default)

► PATHDISP

An optional keyword to specify the disposition of a z/OS UNIX file when the job step ends normally or abnormally. The valid values are:

- KEEP.
- DELETE.

The MVS system default is KEEP,KEEP

► PATHOPTS

An optional keyword to specify the access and status for the z/OS UNIX file named in the PATH parameter. The valid values are:

- ORDONLY: The utility should open the file for read access only.
- OCREAT: If the file does not exist, the system creates it.
- OWRONLY: The utility should open the file for write access only.
- ONONBLOCK: Specifies the following items, depending on the type of file:

For a FIFO special file, such as a UNIX System Services pipe file:

- With ONONBLOCK specified and ORDONLY access: An open() function for reading-only returns without delay.
- With ONONBLOCK not specified and ORDONLY access: An open() function for reading-only blocks (waits) until a process opens the file for writing.
- With ONONBLOCK specified and OWRONLY access: An open() function for writing-only returns an error if no process currently has the file open for reading.
- With ONONBLOCK not specified and OWRONLY access: An open() function for writing-only blocks (waits) until a process opens the file for reading.

For a character special file that supports non-blocking open:

- If ONONBLOCK is specified: An open() function returns without blocking (waiting) until the device is ready or available. the device response depends on the type of device.
- If ONONBLOCK is not specified: An open() function blocks (waits) until the device is ready or available. Specification of ONONBLOCK has no effect on other file types.

The default for LOAD is ORDONLY. The default for UNLOAD is OCREAT and OWRONLY.

For details about these options, see *z/OS V1R11.0 TSO/E Command Reference*, SA22-7782.

### 3.2.9 Recommendations for the use of templates

Templates should be used whenever possible, and used in conjunction with LISTDEF, which provides a powerful mechanism for controlling the housekeeping of a DB2 subsystem. Templates can be used without LISTDEFs where LISTDEF is not supported, for example, copying of catalog and directory.

You can:

- ▶ Use variables to make data set names meaningful, for example, use &LR and &IC to identify image copy data sets.
- ▶ Use PDS members as input to reduce maintenance.
- ▶ Define templates twice, once for tape (where applicable) and once for disk, to allow for quicker interchanges between devices.
- ▶ Use the STACK option for tape data sets.
- ▶ Use PREVIEW mode to review the output from LISTDEF and TEMPLATE.
- ▶ Use the GDG option to ensure that GDGs are utilized consistently.

### 3.2.10 Current restrictions in the use of templates

The variables that can be used with templates are those listed in Table 3-2 on page 50.

Column functions are not supported in templates, for example, SUBSTR.

Data sets whose size is calculated as too large for the DYNALLOC interface must be allocated using DD cards. Message DSNU1034I is issued if this condition is met.

## 3.3 Combining wildcards and templates

The use of LISTDEF often requires the use of TEMPLATE. If the requirement is for a utility to process a dynamic list of DB2 objects, that is, a list which may increase or decrease over time, you can use LISTDEF. As a result, many online utilities require data set allocations for each object processed. Therefore, the number (and parameters) of these allocations must be dynamic too. This can be implemented with TEMPLATE, thus supporting a dynamic number of data set allocations that fit the object or the list of objects being processed by a utility.



For example, primary and backup copies are required per partition for all partitioned table spaces in the database DBX whose names start with PT. This leads to the LISTDEF statement shown in Figure 3-5. When you do not know how many table spaces exist that match that list definition, and not knowing how many partitions these table spaces have, use the TEMPLATE for dynamically allocating the needed copy data sets. As the TEMPLATE includes the variable &PRIBAC, this TEMPLATE can be used in place of both DD names, the one for the primary copy and the one for the backup copy.

TEMPLATE and LISTDEF combined	
<b>Not Using LISTDEF</b>	<pre>//COPYPRI1 DD DSN=DBX.PTS1.P00001.P.D2000199, //          DSIP=...,UNIT=...,SPACE=... //COPYPRI2 DD DSN=DBX.PTS1.P00002.P.D2000199, //          DSIP=...,UNIT=...,SPACE=... //COPYPRI3 DD DSN=DBX.PTS1.P00003.P.D2000199, //          DSIP=...,UNIT=...,SPACE=... //COPYSEC1 DD DSN=DBX.PTS1.P00001.B.D2000199, //          DSIP=...,UNIT=...,SPACE=... //COPYSEC2 DD DSN=DBX.PTS1.P00002.B.D2000199, //          DSIP=...,UNIT=...,SPACE=... //COPYSEC3 DD DSN=DBX.PTS1.P00003.B.D2000199, //          DSIP=...,UNIT=...,SPACE=... //SYSIN DD * COPY TABLESPACE DBX.PTS1 DSNUM 1 COPYDDN (COPYPRI1,COPYSEC1) COPY TABLESPACE DBX.PTS1 DSNUM 2 COPYDDN (COPYPRI2,COPYSEC2) COPY TABLESPACE DBX.PTS1 DSNUM 3 COPYDDN (COPYPRI3,COPYSEC3) /*</pre>
<b>Using LISTDEF</b>	<pre>/* none of the DD statements above //SYSIN DD * LISTDEF COPYLIST INCLUDE TABLESPACE DBX.PT* PARTLEVEL TEMPLATE COPYTMPL DSN ( &amp;DB..&amp;TS..P&amp;PART..&amp;PRIBAC..D&amp;JDATE. ) COPY LIST COPYLIST COPYDDN ( COPYTMPL,COPYTMPL ) /*</pre>

Figure 3-5 TEMPLATE and LISTDEF combined (V6 shown for comparison)

When the job executes, DB2 first processes the LISTDEF statement, then the TEMPLATE definition is read and then stored. When DB2 has read the COPY statement, the list COPYLIST is generated and DB2 copies each item in the list, assuming that there are no restrictions. At this time, DB2 knows all the details for the variables and can substitute them into the TEMPLATE variables stored previously.

With DB2 9, the dynamic allocations are executed when the utility is processing the objects for which the allocation is needed.

If several executions of the CHECK INDEX, REBUILD INDEX, and RUNSTATS INDEX utilities are invoked for multiple indexes for a list of objects, performance can be improved. For example, when you have two table spaces with three indexes each, consider this code for the LISTDEF:

```
LISTDEF myindexes INCLUDE INDEXSPACES TABLESPACE db1.ts1
INCLUDE INDEXSPACES db2.ts2
```

DB2 generates CHECK INDEX statements for all indexes of the same table space as follows:

```
CHECK INDEX db1.ix1,db1.ix2,db1.ix3
CHECK INDEX db2.ix1,db2.ix2,db2.ix3
```

Thus, the table space scans are reduced to two instead of a possible six.

### 3.3.1 Compatibility with utilities

TEMPLATE and LISTDEF can be used according to the table shown in Figure 3-6. Templates can only be used where an output data set is required by the utility, such as COPY, but not MODIFY.

Object Wildcard and Dynamic Allocation by Utility		
Utility	LISTDEF	TEMPLATE
CHECK DATA	No	Yes
CHECK INDEX	Yes	Yes
CHECK LOB	No	Yes
COPY	Yes	Yes
COPYTOCOPY	Yes	Yes
LOAD	No	Yes
MERGECOPY	Yes	Yes
MODIFY RECOVERY	Yes	n/a
MODIFY STATISTICS	Yes	n/a
QUIESCE	Yes	n/a
REBUILD INDEX	Yes	Yes
RECOVER	Yes	n/a
REORG INDEX	Yes	Yes
REORG TABLESPACE	Yes	Yes
REPORT	Yes	n/a
RUNSTATS	Yes	n/a
UNLOAD	Yes	Yes

Figure 3-6 Utility/LISTDEF/TEMPLATE cross-reference

### 3.3.2 Mixing the old and the new

In some cases, you have to use a mixture of the old and the new method of processing. For example, the DB2 catalog and directory cannot be processed via the LISTDEF command using wildcards; they have to be coded explicitly.

The use of LISTDEF and TEMPLATEs go hand-in-hand. The ability to generate lists gives rise to the need for dynamic allocation, so the use of both together is recommended, when their use is supported by the utility (see Figure 3-6). However, this does not mean that you cannot use them independently. For example, if naming standards do not lend themselves to the use of LISTDEF, then TEMPLATEs can still be used to provide dynamic allocations.

Example 3-17 provides an example of using the UNLOAD utility.

*Example 3-17 Using LISTDEF and TEMPLATES together*

---

```
TEMPLATE ULDDDN
  DSN(DB2V710G.&DB..&TS..UNLOAD)
  UNIT(SYSDA) SPACE(45,45) CYL DISP(NEW,CATLG,DELETE)
  VOLUMES(SBOX57,SBOX60)
TEMPLATE PNHDDN
  DSN(DB2V710G.&DB..&TS..PUNCH)
  UNIT(SYSDA) CYL DISP(NEW,CATLG,DELETE)
  VOLUMES(SBOX57,SBOX60)
UNLOAD TABLESPACE U9G01T11.TSCUST
  FROMCOPY DB2V910G.U9G01T11.TSCUST.D2001145.T022853L
  PUNCHDDN PNHDDN UNLDDN ULDDDN
  FROM TABLE PAOLOR1.CUSTOMER
    (C_CUSTKEY,C_NAME,C_ADDRESS)
  WHEN ( C_CUSTKEY > 1000 )
```

---

### 3.3.3 Object in restricted status

When a DB2 object is in a restricted state, for example, STOP, that prevents a utility from executing successfully, this is *not* recognized during LISTDEF processing. The LISTDEF and TEMPLATE processing is executed normally, for example, lists are built and TEMPLATES are constructed. When the utility processes the restricted object, it will produce an Unavailable Resource message. The actions of the utility then depend upon the OPTIONS control statement. If OPTIONS(ITEMERROR,SKIP) is specified, processing continues with the next object and will set a return code of 8. If this OPTION is not specified, then processing stops at the failing object, again with a return code of 8. If the utility running would normally terminate with an abend in this situation, ITEMERROR will not convert the abend to a RC8.

The benefits of using LISTDEF and TEMPLATES in this situation are that data sets are only allocated at the time that they are required, so in the above scenario, there are no extraneous data sets to delete. If TEMPLATES are not being used, then it is the user's responsibility to ensure that the data sets are managed correctly.

### 3.3.4 Utility restartability

When restarting a utility job stream that utilizes LISTDEF and TEMPLATES, restartability within the job is controlled via a checkpoint list built from the LISTDEF, and is stored in SYSUTILX. The LISTDEF is not reevaluated again. DB2 will automatically alter the disposition of the required data set to that required for a restart, for example, using REORG (see Figure 3-3 on page 57).

Default values for disposition vary depending upon the utility and whether the utility is restarting. Default values are shown respectively in Table 3-4 and Table 3-5 on page 71.

*Table 3-4 Data disposition for dynamically allocated data sets*

ddname	CHECK DATA	CHECK INDEX OR CHECK LOB	COPY	LOAD	MERGE COPY	REBUILD INDEX	REORG INDEX	REORG TABLE SPACE	UNLOAD
SYSREC	Ignored	Ignored	Ignored	OLD, KEEP, KEEP	Ignored	Ignored	Ignored	NEW, CATLG, CATLG	NEW, CATLG, CATLG
SYSDISC	Ignored	Ignored	Ignored	NEW, CATLG, CATLG	Ignored	Ignored	Ignored	NEW, CATLG, CATLG	Ignored
SYSPUNCH	Ignored	Ignored	Ignored	Ignored	Ignored	Ignored	Ignored	NEW, CATLG, CATLG	NEW, CATLG, CATLG
SYSCOPY	Ignored	Ignored	NEW, CATLG, CATLG	NEW, CATLG, CATLG	NEW, CATLG, CATLG	Ignored	Ignored	NEW, CATLG, CATLG	Ignored
SYSCOPY2	Ignored	Ignored	NEW, CATLG, CATLG	NEW, CATLG, CATLG	NEW, CATLG, CATLG	Ignored	Ignored	NEW, CATLG, CATLG	Ignored
SYSRCPY1	Ignored	Ignored	NEW, CATLG, CATLG	NEW, CATLG, CATLG	NEW, CATLG, CATLG	Ignored	Ignored	NEW, CATLG, CATLG	Ignored
SYSRCPY2	Ignored	Ignored	NEW, CATLG, CATLG	NEW, CATLG, CATLG	NEW, CATLG, CATLG	Ignored	Ignored	NEW, CATLG, CATLG	Ignored
SYSUT1	NEW, DELETE, CATLG	NEW, DELETE, CATLG	Ignored	NEW, DELETE, CATLG	Ignored	NEW, DELETE, CATLG	NEW, DELETE, CATLG	NEW, DELETE, CATLG	Ignored
SORTOUT	NEW, DELETE, CATLG	Ignored	Ignored	NEW, DELETE, CATLG	Ignored	Ignored	NEW, DELETE, CATLG	NEW, DELETE, CATLG	Ignored
SYSMAP	Ignored	Ignored	Ignored	NEW, CATLG, CATLG	Ignored	Ignored	Ignored	Ignored	Ignored
SYSERR	NEW, CATLG, CATLG	Ignored	Ignored	NEW, CATLG, CATLG	Ignored	Ignored	Ignored	Ignored	Ignored

Table 3-5 Data dispositions for dynamically allocated data sets on RESTART

ddname	CHECK DATA	CHECK INDEX OR CHECK LOB	COPY	LOAD	MERGE COPY	REBUILD INDEX	REORG INDEX	REORG TABLE SPACE	UNLOAD
SYSREC	Ignored	Ignored	Ignored	OLD, KEEP, KEEP	Ignored	Ignored	Ignored	MOD, CATLG, CATLG	MOD, CATLG, CATLG
SYSDISC	Ignored	Ignored	Ignored	MOD, CATLG, CATLG	Ignored	Ignored	Ignored	MOD, CATLG, CATLG	Ignored
SYSPUNCH	Ignored	Ignored	Ignored	Ignored	Ignored	Ignored	Ignored	MOD, CATLG, CATLG	MOD, CATLG, CATLG
SYSCOPY	Ignored	Ignored	MOD, CATLG, CATLG	MOD, CATLG, CATLG	MOD, CATLG, CATLG	Ignored	Ignored	MOD, CATLG, CATLG	Ignored
SYSCOPY2	Ignored	Ignored	MOD, CATLG, CATLG	MOD, CATLG, CATLG	MOD, CATLG, CATLG	Ignored	Ignored	MOD, CATLG, CATLG	Ignored
SYSRCPY1	Ignored	Ignored	MOD, CATLG, CATLG	MOD, CATLG, CATLG	MOD, CATLG, CATLG	Ignored	Ignored	MOD, CATLG, CATLG	Ignored
SYSRCPY2	Ignored	Ignored	MOD, CATLG, CATLG	MOD, CATLG, CATLG	MOD, CATLG, CATLG	Ignored	Ignored	MOD, CATLG, CATLG	Ignored
SYSUT1	MOD, DELETE, CATLG	MOD, DELETE, CATLG	Ignored	MOD, DELETE, CATLG	Ignored	MOD, DELETE, CATLG	MOD, CATLG, CATLG	MOD, DELETE, CATLG	Ignored
SORTOUT	MOD, DELETE, CATLG	Ignored	Ignored	MOD, DELETE, CATLG	Ignored	Ignored	MOD, DELETE, CATLG	MOD, DELETE, CATLG	Ignored
SYSMAP	Ignored	Ignored	Ignored	MOD, CATLG, CATLG	Ignored	Ignored	Ignored	Ignored	Ignored
SYSERR	MOD, CATLG, CATLG	Ignored	Ignored	MOD, CATLG, CATLG	Ignored	Ignored	Ignored	Ignored	Ignored

### 3.3.5 Use with partitions

Partitions must be specified slightly differently from non-partitioned table spaces. If the utility processes at the partition level, then first the table space (DSNUM 0) is EXCLUDED and then the partitions are INCLUDED using PARTLEVEL.

## 3.4 OPTIONS

The utility control statement **OPTIONS** is provided to complement the functionality offered by **LISTDEF** and **TEMPLATE**s. **OPTIONS** provide a general mechanism to enter options that affect utility processing across multiple utility executions in a job step. Figure 3-7 illustrates the functions that can be utilized by specifying the respective keywords in the **OPTIONS** statement. **OPTIONS** should be coded within the **SYSIN** in front of the utility control statements to which the options apply. Any subsequent **OPTIONS** statements entirely supersede any prior statements.

### Several other functions: **OPTIONS**

Function	<b>OPTIONS</b> keyword [parameter]
Test feature	PREVIEW
Identify / override default DD names:	
- for list definitions	LISTDEFDD dd-name
- for templates	TEMPLATEDD dd-name
Error handling	EVENT ( ITEMERROR, HALT ! SKIP )
Handling of warning messages	EVENT ( WARNING, RC0 ! RC4 ! RC8 )
Restore default options	OFF
and:	
Utility activation key	KEY

Example: **OPTIONS LISTDEFDD MYLISTS EVENT ( ITEMERROR, SKIP, WARNING, RC8 )**  
**COPY LIST COPYLIST COPYDDN ( COPYTMPL, COPYTMPL )**

Reset rule: An **OPTIONS** statement replaces any prior **OPTIONS** statement

Figure 3-7 *OPTIONS* functions

### 3.4.1 **OPTIONS(PREVIEW)**

Specifying **OPTIONS(PREVIEW)** will expand the list definitions and the **TEMPLATE** definitions and report their contents whenever a utility statement, for example, **COPY**, follows. All utility control statements are parsed for syntax errors, but normal utility execution will not take place. If the syntax is valid, all **LISTDEF** lists and **TEMPLATE** data set names that appear in **SYSIN** will be expanded and the results printed to the **SYSPRINT** data set. A useful feature of this process is that the expanded list from the **SYSPRINT** data set can be captured and used inside a utility control statement to avoid the repetition of the expansion process. Lists from the **SYSLISTD** DD data set and **TEMPLATE** data set names from the **SYSTEMPL** DD data set that are referenced by a utility invocation will also be expanded.

**OPTIONS(PREVIEW)** does not check the validity of the **TEMPLATE** data set names, their syntax, or whether the data sets names are unique.

For a preview example, see the job output in Example 3-5 on page 54 showing the **LISTDEF** expansion.

The option `OPTIONS(PREVIEW)` is identical to the `PREVIEW EXEC` parameter (see Figure 3-8), which turns on the preview mode for the whole job step. With the new `OPTIONS(PREVIEW)` utility control statement, you have a more detailed granularity: Preview mode, via the utility control statement, can be switched on and off multiple times within a single job step, but when preview mode has been initiated by the JCL parameter `PREVIEW`, the `OPTIONS` control statement cannot switch it off.

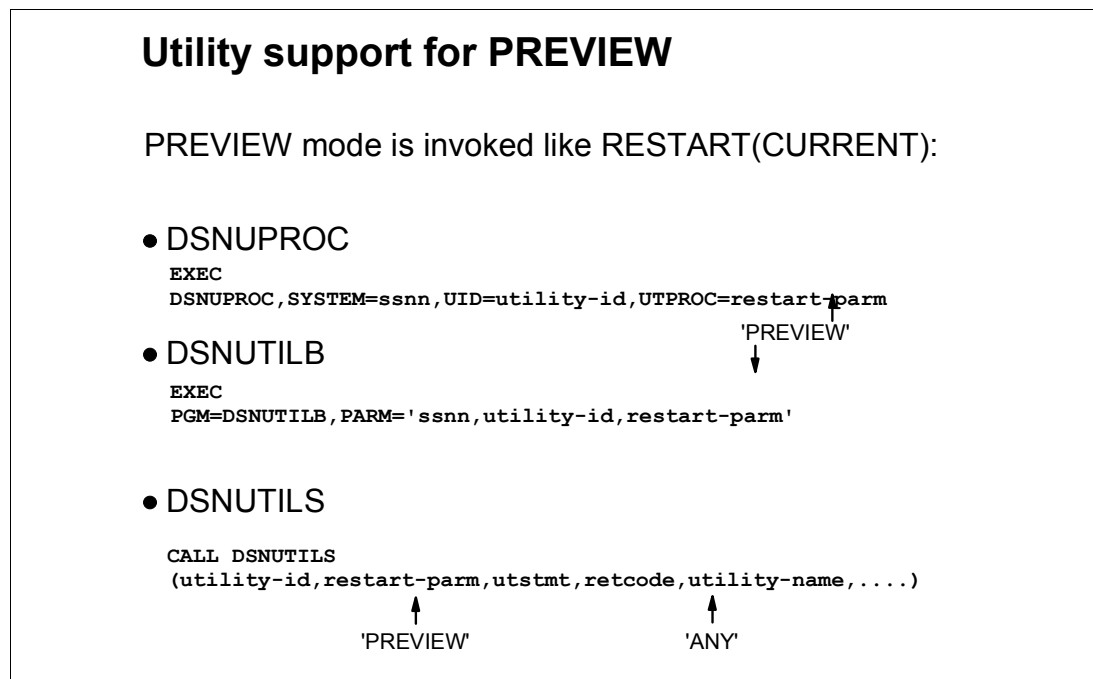


Figure 3-8 Utility support for *PREVIEW*

`PREVIEW` can also be initiated through `DSNUTILS`, and the use of “ANY” suppresses all dynamic allocations done by `DSNUTILS`. Using `PREVIEW`, this list can be saved before the actual utility is run, and if there is an error, the job output will show the list item where it failed. Since `DSNUTILS` does not provide for storing the reduced list, this saved list can be edited by removing the already processed items and starting again with this reduced list.

### 3.4.2 Other `OPTIONS` functionality

`OPTIONS` can also be used in the following circumstances:

- It can override default ddnames for `LISTDEF` and `TEMPLATES` by using the following syntax:
 

```
OPTIONS LISTDEFDD(listindd) TEMPLATEDD(tempindd)
```
- DB2 handling of errors can be controlled; this can only be used when processing a list.
  - To halt on such errors during list processing (default), use the following syntax:
 

```
OPTIONS EVENT(ITEMERROR, HALT)
```
  - To skip any error and keep going, use the following syntax:
 

```
OPTIONS EVENT(ITEMERROR, SKIP)
```

- Handling Return code 4 from a utility is available whether using a list or not by using the following syntax:

```
OPTIONS EVENT(WARNING,RC0)
OPTIONS EVENT(WARNING,RC4)
OPTIONS EVENT(WARNING,RC8)
```

- You can restore the defaults by using the following syntax:

```
OPTIONS OFF
```

**Note:** No other keywords may be specified when using the OPTIONS OFF setting.

OPTIONS OFF is equivalent to:

```
OPTIONS LISTDEFDD SYSLISTD TEMPLATEDD SYSTEMPL
        EVENT (ITEMERROR, HALT, WARNING, RC4)
```

As usual for utility control statements, parentheses are required for a list, in contrast to a single item, to be specified.

## 3.5 Using Unicode control statements

All utility input statements can be written in EBCDIC or Unicode. DB2 typically reads utility control statements from the SYSIN data set. DB2 can read LISTDEF control statements from the SYSLISTD data set and TEMPLATE control statements from the SYSTEMPL data set. All these statements can be coded in Unicode UTF-8 (code page 1208). All control statements in a given data set must be written entirely in the same character set.

DB2 automatically detects and processes Unicode UTF-8 control statements if the first character of the data set is:

- A Unicode UTF-8 blank (x'20')
- A Unicode UTF-8 dash (x'2D')
- A Unicode UTF-8 upper case A through Z (x'41' through x'5A')

In all other cases, the control statement data set is processed as EBCDIC. As shown in Example 3-18, an informational message is issued to identify the character set that is being processed.

*Example 3-18 Identify the character set used in the utility control statements*

---

```
DSNU1045I  DSNUGTIS - PROCESSING SYSIN AS UNICODE
```

---

Utility control statements submitted in UNICODE are translated into EBCDIC before processing; however, character string constants are not translated. Character string constants are left in the character set in which they were specified. In some cases, it may be necessary to use hexadecimal string constants in order to achieve the desired behavior.





## Performance via parallelism of executions

In each new version of DB2, IBM has improved the performance of the utilities. This addresses the needs of customers who are challenged with increasing amounts of data and a shrinking batch window.

In this chapter, we focus on parallelism, which is one of the techniques that DB2 uses to shorten the elapsed times of utilities. Maximizing the exploitation of parallelism is of primary importance in reducing the elapsed time for utilities.

We also discuss the use of the keyword SORTKEYS, which is used for controlling Parallel Index Build (PIB) and the possible constraints that can reduce the level of parallelism in utilities.

Note that we cover only those aspects that are common for the different utilities. For more details and examples, we refer to the chapters on the individual utilities.

This chapter contains the following sections:

- ▶ Levels of parallelism
- ▶ Inline executions
- ▶ COPY and RECOVER of a list of DB2 objects
- ▶ Parallelism with the LOAD utility
- ▶ Partition parallelism with the UNLOAD utility
- ▶ Parallelism with the REORG TABLESPACE utility
- ▶ Parallelism with the REBUILD INDEX utility
- ▶ Parallelism with the CHECK INDEX utility
- ▶ Considerations for running parallel subtasks

## 4.1 Levels of parallelism

Different forms of parallelism exist in the utilities. Consider the following:

- ▶ Inline execution, which is the execution of different types of utilities in parallel subtasks instead of executing them serially. Examples are COPY and RUNSTATS running embedded and in parallel with LOAD and REORG utilities.
- ▶ Execution of a utility on different DB2 objects in parallel. Examples are COPY/RECOVER of a list of table spaces in parallel or rebuilding the different indexes of a table space in parallel during LOAD or REORG TABLESPACE. The latter is called Parallel Index Build or PIB.
- ▶ Partition Parallelism when executing a utility on partitioned objects. DB2 uses parallel subtasks to load and unload partitions in parallel and for the index build processing. Examples are rebuilding the parts of a partitioned index in parallel during REBUILD INDEX, unloading the partitions of a partitioned table space in parallel during UNLOAD or REORG TABLESPACE, and loading the partitions of a partitioned table space in parallel during LOAD or REORG TABLESPACE.

Other forms of parallelism also exist, such as Log Apply parallel tasks during the log phase of REORG TABLESPACE PARTx SHRLEVEL REFERENCE or CHANGE in DB2 9, where log records are applied in parallel on the shadows of the non-partitioned indexes (as a result of the functional change after the elimination of the BUILD2 phase).

In the past, some of this parallelism could also be achieved by the users by submitting different jobs in parallel, sometimes ending up with considerable contentions. Today, all these parallel executions can be done in one job, using parallel subtasks.

## 4.2 Inline executions

The term *inline execution* refers to the execution of different types of utilities in parallel subtasks instead of executing them serially. Examples are COPY and RUNSTATS running embedded and in parallel with LOAD and REORG utilities.

When executing utilities, such as REORG or LOAD of a table space, without logging, the DB2 object is left in a restrictive state, COPY pending, and an image copy is required to make the object fully available to the applications. To ensure that the correct access paths are chosen, the RUNSTATS utility is also recommended to be run to update the statistics of the table. These utilities add to the time and resources (CPU and I/O) required to carry out the original utility. DB2 addressed these issues with the introduction of the Inline COPY and RUNSTATS functionality. These enhancements ensure, when the original utility completes, that the object are fully available to the application, and that the statistics are current.

We recommend using inline copy and inline statistics instead of running separate COPY and RUNSTATS utilities.

**Note:** In DB2 9, it is possible to LOAD a table space with the NOCOPYPEND option to avoid having the table space put into the COPY pending restrictive state, even if there is no good image copy available afterwards. Because this table space cannot be recovered, it can only be used for data that can be easily recreated. See 9.8, “NOCOPYPEND option” on page 143 for more details.

## 4.2.1 Inline COPY

LOAD REPLACE and REORG TABLESPACE have the ability to take an image copy while loading or reloading data into a table, as shown in Example 4-1. This leads to the table space *not* being put into COPY pending status even if LOG NO is specified. Up to four copies are allowed (two primary and two recovery) and the process is initiated by the use of the COPYDDN/RECOVERYDDN control statements, as detailed in Example 4-1. All image copies taken are taken as a full SHRLEVEL REFERENCE copy.

*Example 4-1 Initiating multiple inline COPY within LOAD utility*

---

```
LOAD DATA REPLACE COPYDDN ddname1,ddname2 RECOVERYDDN dname3,ddname4 INTO TABLE tname
```

---

```
REORG TABLESPACE dbname.tsname COPYDDN ddname1,ddname2 RECOVERYDDN dname3,ddname4
```

---

Example 4-2 shows the Inline COPY at partition level.

*Example 4-2 Initiating inline COPY when loading at a partition level*

---

```
LOAD DATA INTO TABLE tname PART 2 REPLACE COPYDDN ddname1
```

---

```
REORG TABLESPACE dbname.tsname PART 2 COPYDDN ddname1
```

---

**Note:** If COPYDDN is specified without the REPLACE option, the LOAD is terminated.

Although DB2 supports image copies of indexes, DB2 does not support the concept of Inline COPY of an index. Not having an image copy of an index defined with the COPY YES attribute will put the index in the ICOPY state, but in contrast with the COPYP status, this is not a restrictive state.

The inline image copy data sets are not exactly the same as the data sets produced by the COPY utility:

- ▶ They are logically the same, except that:
  - Data pages may be out of sequence and some might be repeated. If repeated, the last occurrence is the correct page.
  - Space map pages may be out of sequence and repeated.
  - A compression dictionary, if being built, is recorded twice. The second one is the correct one.
  - Normally, the total number of duplicate pages is a small percentage, with a negligible effect on the required space for the data set.
- ▶ SYSCOPY is updated with ICTYPE = F and SHRLEVEL = R. The originating utility can be found in STYPE (R for LOAD REPLACE LOG YES, S for LOAD REPLACE LOG NO, W for REORG LOG NO, and X for REORG LOG YES). If STYPE is blank or contains another value, the image copy is not an Inline COPY.
- ▶ DSN1COPY and DSN1PRNT require extra options to process inline copies. Refer to “Using inline copies with DSN1COPY and DSN1PRNT” on page 79.
- ▶ Inline COPY data sets from the LOAD utility are *not* recommended for use with RECOVER TOCOPY, because they are taken during the LOAD phase before any DISCARD phases; therefore, copies may contain unique index or RI violations. They can be used in point-in-time recovery. If a RECOVER TOCOPY is executed, then all indexes are placed in rebuild pending status, and dependent table spaces, if any, are placed in check pending status. REBUILD INDEX and CHECK DATA must be run.

- If you take an inline image copy of a table space with a table that has LOB or XML columns, DB2 makes a copy of the base table space, but does not copy the LOB or XML table spaces.
- REORG SHRLEVEL REFERENCE can take inline image copies of LOB and XML table spaces when reorganizing them.

The benefits of taking an inline image copy is that the elapsed time is reduced when compared with running the utility followed by a copy, and that the time unavailable to the applications could also be shorter. Considerable savings can be made in elapsed time, while consuming virtually the same amount of CPU time, by using Inline COPY.

## Using inline copies

Although inline copies are recorded in SYSCOPY as full image copies with SHRLEVEL REFERENCE, they are different from normal image copies because they can contain more pages, as well as duplicate pages.

Inline copies produced during the LOAD phase of the LOAD utility can contain data that was removed afterwards during the INDEXVAL and ENFORCE phases (duplicates and RI violating rows).

During the LOG phase of REORG, incremental image copies are taken. These incremental copies are appended to the image copy data set created during the RELOAD phase. The full image copy, which results from a successful Online REORG, will probably contain duplicate pages, and therefore it will be bigger than other full image copies of this table space. The later pages will contain the most recent changes.

The DB2 RECOVER utility will take this into account and apply the most recent pages, but there are some restrictions regarding the stand-alone utilities DSN1COPY and DSN1PRNT.

### ***Using inline copies with the RECOVER utility***

As previously stated, inline copies created during LOAD should not be used in a RECOVER TOCOPY scenario. The RECOVER utility handles the inline copies differently from “normal” copies.

In an Inline COPY taken by the REORG utility, there is a full image copy and multiple incremental copies in the data set. RECOVER processes the data pages in order, and any duplicate pages are replaced as the utility progresses. As the incremental image copies contain any change pages, these are used to replace the corresponding pages in the table spaces. Thus, at the end of the table, spaces are correctly recovered.

For an Inline COPY taken with the LOAD REPLACE LOG NO option, the image copy is taken during the LOAD phase of the utility. This means that any rows discarded during the INDEXVAL or ENFORCE stage are included in the copy. Recovery to a point in time applies the image copy and then has to process the discarded rows. If the Inline COPY is taken at time T1, and then discard processing removes a row at time T2, RECOVER ensures that if a recovery to time T3 is required, then the row discarded is not present when the recovery is complete. The method of ensuring data integrity is that the discard processing is logged even when LOG NO is specified. This ensures that data integrity is not compromised.

### ***Using inline copies with the UNLOAD utility***

If an image copy, used as input to the UNLOAD utility, was created by an inline copy operation (LOAD or REORG TABLESPACE), the image copy can contain duplicate pages. If duplicate pages exist, the UNLOAD utility issues a warning message, and all the qualified rows in duplicate pages are unloaded into the output data set.

### ***Using inline copies with DSN1COPY and DSN1PRNT***

As described above, the Inline COPY produced during REORG is different from a normal full image copy. To avoid error messages during DSN1COPY or DSN1PRNT processing, use the parameter INLCOPY to specify that the input data set is an inline copy instead of the FULLCOPY parameter.

For a compressed table space, DSN1COPY does not reset the dictionary version for an inline image copy. To reset the dictionary version for an inline image copy, use the inline image copy as input to DSN1COPY with a VSAM intermediate data set as output. This intermediate data set can then be used as input to DSN1COPY RESET to copy the intermediate data set to the real target data set. The dictionary version is used to check the consistency between the dictionary present in the table space and the compressed rows. It is a RBA or LRSN value and should be reset when copying data to another DB2 subsystem.

## **4.2.2 Inline RUNSTATS**

Inline statistic gathering can be used with the LOAD, REORG TABLESPACE, REORG INDEX, and REBUILD INDEX utilities. This feature allows for the collection of statistics to be taken during the LOAD and BUILD phases of the utility, thereby eliminating the need for the execution of a separate RUNSTATS utility after the utility finishes.

The main objective of inline statistics is to reduce the total elapsed time of a REORG, LOAD, or RECOVER/REBUILD index followed by statistics gathering. This is achieved by the elimination of additional scans of the data for gathering the statistics. The collection is initiated by the inclusion of the STATISTICS keyword, and all RUNSTATS options are supported, including the HISTORY functionality.

These are some restrictions for inline statistics gathering:

- ▶ LOAD with Inline statistics is only valid for REPLACE or RESUME NO options.
- ▶ Table space statistics collected during REORG TABLESPACE SHRLEVEL CHANGE do not reflect the changes to the originating table space after the UNLOAD phase. If a large number of updates are expected during the Log Apply phase, and accurate statistics are required, then it is better to run RUNSTATS afterwards.
- ▶ Index statistics gathered during REORG INDEX SHRLEVEL CHANGE do not reflect the changes after the BUILD phase.
- ▶ If you restart a utility job that uses the STATISTICS keyword, in most cases inline statistics collection does not occur. To update catalog statistics, run the RUNSTATS utility afterwards.
- ▶ Rows discarded during the LOAD process may or may not be reflected in the statistics. If a large number of discards are expected and accurate statistics are required, then the use of inline statistics is not recommended. A RUNSTATS utility should be run afterwards instead.
- ▶ If you collect statistics of a table that has LOB or XML columns, statistics are collected on the base table space, not on the LOB or XML table spaces.
- ▶ If you specify a table for which column statistics are to be collected, the table cannot be a table CLONE.

Collecting statistics is explained at Chapter 11, “Gathering statistics” on page 291.

## 4.3 COPY and RECOVER of a list of DB2 objects

Another type of parallelism in DB2 utilities is the execution of one type of utility on different DB2 objects in *parallel subtasks*. In this section, we discuss the COPY and RECOVER of a list of table spaces in parallel.

You can copy or recover a list of objects in parallel to improve performance. Specifying a list of objects along with the COPY SHRLEVEL REFERENCE option creates a single recovery point for that list of objects. Specifying the PARALLEL keyword allows you to copy a list of objects in parallel, rather than serially. When COPY and RECOVER process objects in parallel, the objects are sorted by size and the largest objects are processed first. One of the benefits of the list processing is to create a consistent backup for the objects in the list:

- ▶ With COPY SHRLEVEL REFERENCE, DB2 will first drain all writers on all objects of the list before starting the actual copying. All image copies will have the same RBA/LRSN value in SYSIBM.SYSCOPY, and the copy set results in a good recovery point if you need to PIT RECOVER these objects back to this RBA/LRSN.
- ▶ With COPY SHRLEVEL CHANGE, DB2 will drain the writers object by object, and the RBA/LRSN values will be different for each object. The set will not be consistent for PIT recovery.

### 4.3.1 COPY in parallel

An example using COPY PARALLEL(3) with a list of five objects is illustrated in Figure 4-1.

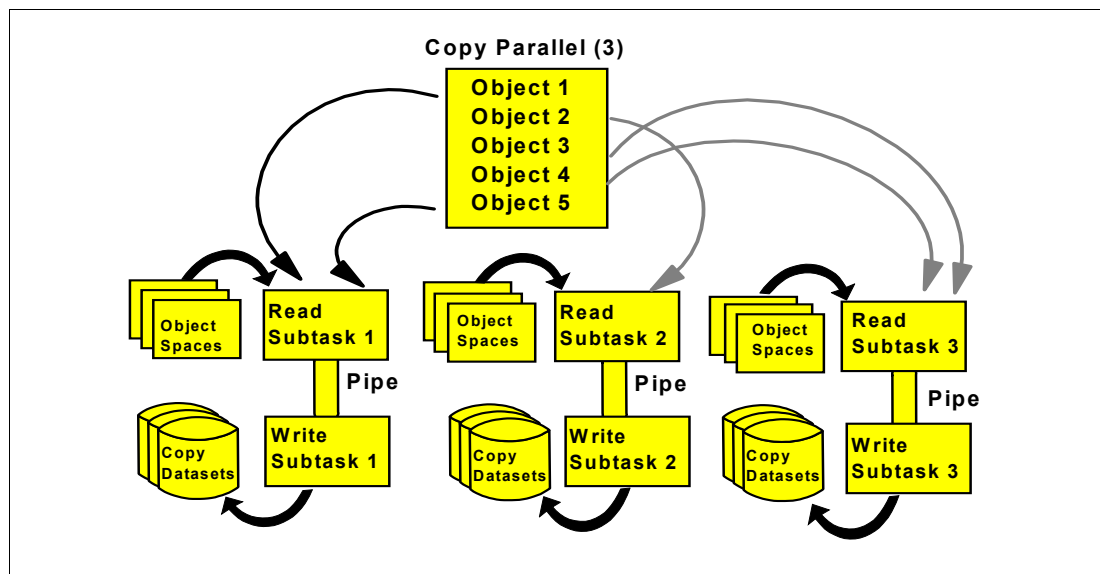


Figure 4-1 Copy a list of DB2 objects in parallel

In this example, COPY creates three subtask sets. Each subtask set consists of two subtasks, one for reading the table/index and one for writing to the image copies. The objects 1, 2, and 3 are assigned to subtask set 1, 2, and 3, respectively. Object 3 finishes first, so object 4 is assigned to subtask set 3. Object 1 finishes next, and object 5 is assigned to subtask set 1. While the read subtask is putting pages into a pipe, the write subtask is pulling them out and writing them to the output data set.

The PARALLEL keyword specifies the maximum number of objects in the list that are to be processed in parallel. The utility processes the list of objects in parallel for image copies being written to different disk or tape devices. If you omit PARALLEL, the list is not processed in parallel. If you specify 0 or do not specify a value, COPY determines the optimal number of objects to process in parallel. DB2 may reduce the number of objects processed in parallel if there is a virtual storage constraint in the utility batch address space.

See *DB2 Version 9.1 for z/OS Utility Guide and Reference*, SC18-9855 for more examples of COPY with the PARALLEL keyword and special considerations when the image copies are written to tape devices.

### 4.3.2 RECOVER in parallel

An example using RECOVER PARALLEL(3) with a list of five objects is illustrated in Figure 4-2.

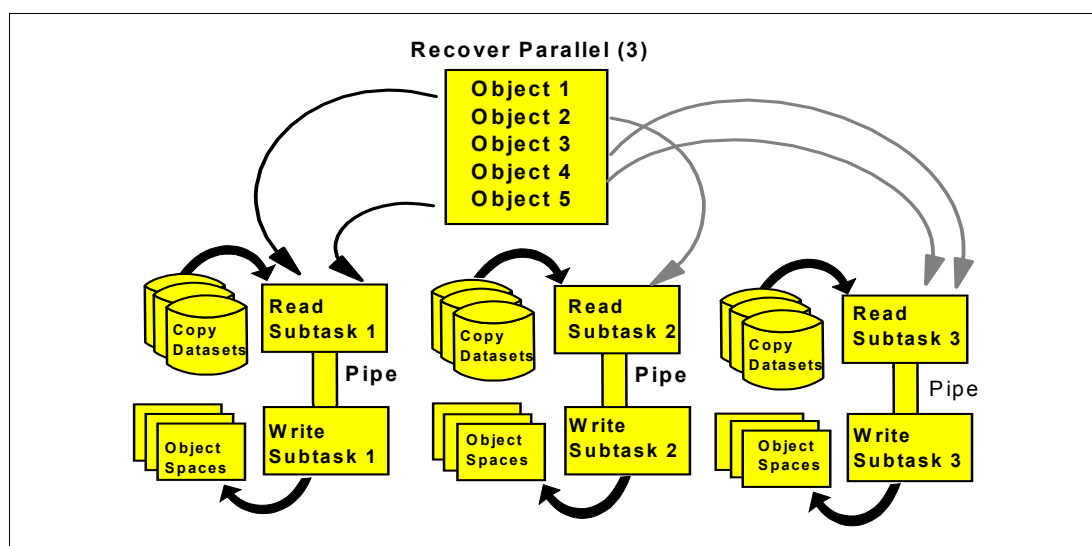


Figure 4-2 Recovering a list of DB2 objects in parallel

In this example, RECOVER creates three subtask sets. A subtask set consists of two subtasks, one for reading the image copies and one for writing to the object space. Objects 1, 2, and 3 are assigned to subtask set 1, 2, and 3, respectively. Object 3 finishes first, so object 4 is assigned to subtask set 3. Then, object 1 finishes next and object 5 is assigned to subtask set 1. While the read subtask is putting pages into the pipe, the write subtask is pulling them out and writing them to the object space(s).

The PARALLEL keyword specifies the maximum number of objects in the list that are to be restored in parallel from image copies on disk or tape. RECOVER attempts to retain tape mounts for tapes that contain stacked image copies when the PARALLEL keyword is specified. In addition, to maximize performance, RECOVER determines the order in which objects are to be restored. PARALLEL also specifies the maximum number of objects in the list that are to be restored in parallel from system-level backups that have been dumped to tape. The processing may be limited by DFSMSHsm. If you specify 0 or do not specify a value, RECOVER determines the optimal number of objects to process in parallel. DB2 may reduce the number of objects processed in parallel if there is a virtual storage constraint in the utility batch address space.

Parallelism is only achieved in the RESTORE phase. The Log Apply processing does not begin until all objects have been restored. During the Log Apply processing, the log will only be scanned once, and recovery will use the fast Log Apply feature, as explained in 13.5, “Fast Log Apply” on page 251.

See *DB2 Version 9.1 for z/OS Utility Guide and Reference*, SC18-9855 for more examples of RECOVER with the PARALLEL keyword and special considerations when restoring image copies from tape or restoring data sets from system-level backups.

## 4.4 Parallelism with the LOAD utility

Parallel Index Build (PIB) enables the LOAD utility to build the indexes of the table space in parallel. Furthermore, if the table space is partitioned, partitions can be loaded in parallel as well. Both of these features will reduce significantly the elapsed time of the LOAD utility.

PIB is activated by default and can be controlled by the SORTKEYS *n* option, where *n* is an estimation of the number of keys to be sorted. If you specify SORTKEYS 0 (zero) or SORTKEYS NO, PIB will not be activated.

The accelerating effects of PIB are as follows:

- ▶ The SORT, the BUILD, and, optionally, the inline STATISTICS phases are performed in parallel. SORT and BUILD are combined in one SORTBLD task.
- ▶ Considerable I/O processing is eliminated by not using SYSUT1 and SORTOUT.

By default, LOAD computes the default SORTKEYS *n* value based upon the number of records in the input data set. It will be zero and PIB consequently disabled when:

- ▶ The target table has only one index or no indexes. In this case, parallelism does not apply.
- ▶ The input is on tape, a cursor, a PDS member, a HFS file, a virtual file, or for SYSREC DD \*. In this case, the utility cannot estimate the number of records to be loaded.

**Tip:** Specify a SORTKEYS value to the LOAD utility to enable PIB when the input data set is on tape or a PDS member. This will also relieve you from the task of adding space estimations to the SORT and Inline COPY data sets.

You can also estimate the SORTKEYS *n* value yourself. To estimate the number of keys to sort:

1. Count 1 for each index.
2. Count 1 for each foreign key where foreign key and index definitions are not identical.
3. For each foreign key where foreign key and index definitions are identical:
  - c. Count 0 for the first relationship in which the foreign key participates.
  - d. Count 1 for subsequent relationships in which the foreign key participates (if any).
4. Multiply the count by the number of rows to be loaded.
5. If more than one table is being loaded, repeat the preceding steps for each table and sum the results.



DB2 uses multiple pairs of sort and build subtasks so that indexes are built in parallel, thereby further improving the elapsed time of the utility. All this is done in the *SORTBLD* phase, which replaces the SORT and BUILD phases. The sort subtask sorts the extracted keys, while the build subtask builds the index. This is also done partially in parallel, because the index build starts as soon as the sort subtask passes the first sorted keys. Moreover, the SYSUT1 and SORTOUT work data sets are not used during the SORTBLD because the key/rid pairs are now directly piped in memory between the subtasks, eliminating considerable I/O processing.

Each sort subtask uses its own sort work data set and its own message data set. If inline statistics are also collected, there is a separate RUNSTATS subtask for each build task that collects the sorted keys and updates the catalog in parallel. When collecting statistics on the table space, an additional subtask runs in parallel with the RELOAD phase.

To activate parallel partition loading for partitioned table spaces, the following rules apply:

- ▶ Separate input data sets have to be supplied per partition.
- ▶ The syntax of the LOAD has to specify each partition to be loaded via the INTO TABLE option, as shown in Figure 4-3.

The need to supply separate input files is made easier with the TEMPLATE functionality described in Chapter 3, “Simplifying utilities with wildcarding and templates” on page 47, and we recommend that this feature be used for loading partitions in parallel.

## LOAD - syntax for parallel partition loading

```
LOAD
  INTO TABLE tb-name PART 1
      INDDN inddn1 DISCARDNN
discddn1
  INTO TABLE tb-name PART 2
      INDDN inddn2 DISCARDNN
discddn2
  ...
  INTO TABLE tb-name PART n
      INDDN inddnn DISCARDNN
discddnn
```

DSNU,DB2I, DSNUTILS support via templates

Figure 4-3 LOAD syntax for activating parallel partition loading

**Note:** Parallel partition loading is not supported for the new partition-by-growth universal table spaces because these can only be loaded at the table space level, not at the partition level.

By using parallel partition LOAD and PIB together, DB2 will initiate subtasks for the LOAD phase and the SORTBLD phase. This is shown in Figure 4-4. For loading partitions in parallel, the optimum is to initiate one subtask per partition to be loaded and two for each index build (sort and build), although this is not always possible due to constraints, such as virtual memory, DB2 threads, and processors available. See 4.9, “Considerations for running parallel subtasks” on page 90 for more details.

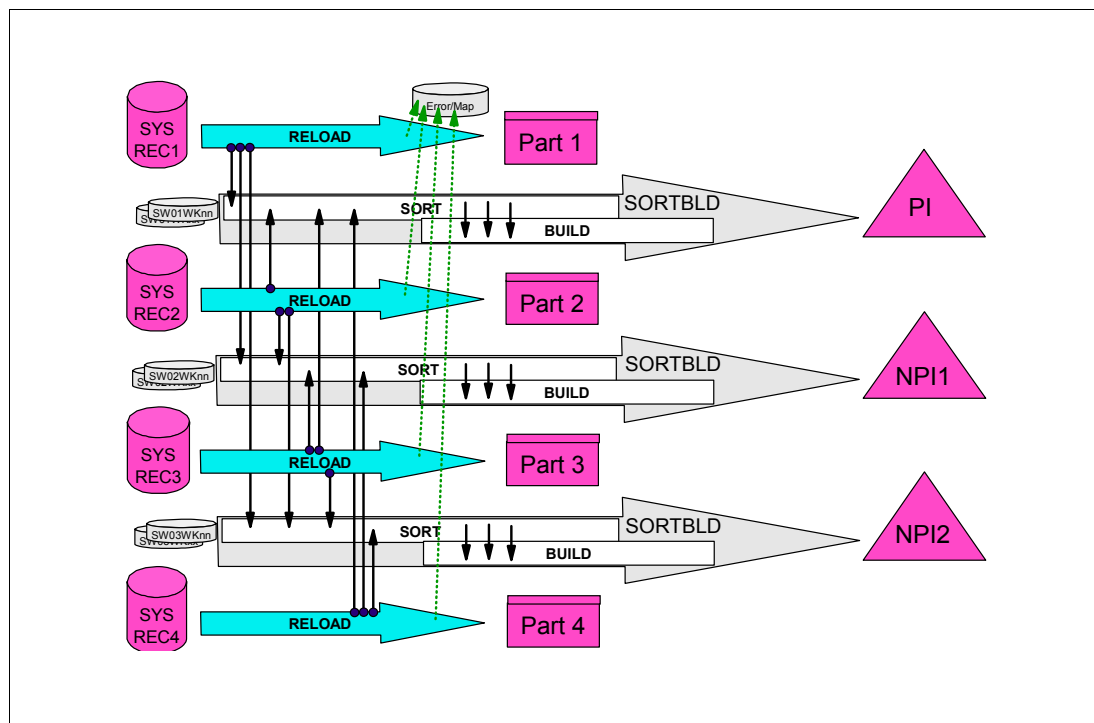


Figure 4-4 Partition parallel LOAD with PIB

Since DB2 7, DB2 enables loading the partitions of a partitioned table in parallel, so dropping the NPIs before the LOAD of the entire table space and rebuilding them afterwards with REBUILD INDEX is no longer beneficial. We recommend changing your existing LOAD jobs if you still have any.

See 7.1.1, “Invoke parallelism” on page 170 for some examples.

## 4.5 Partition parallelism with the UNLOAD utility

With the UNLOAD utility, you can unload a table or table space to a sequential data set. When the table space is partitioned, it is also possible to unload one or more partitions to different sequential data sets.

When unloading a partitioned table space into individual data sets (one per partition), the UNLOAD utility automatically activates multiple parallel subtasks. Optimally, the number of subtasks will be equal to the number of partitions to be unloaded, but most of the time it will be limited to the number of CPUs available.

There are different ways to influence the number of parallel subtasks. See 4.9, “Considerations for running parallel subtasks” on page 90 for details.

The parallelism of the UNLOAD utility is enabled by using the PART option in the UNLOAD statement or using the PARTLEVEL option in the LISTDEF command. See 11.4.2, “Unload by partition and parallelism” on page 212 for more details.

## 4.6 Parallelism with the REORG TABLESPACE utility

As with the LOAD utility, PIB enables the REORG TABLESPACE utility to build the indexes of the table space in parallel. Furthermore, if the table space is partitioned, partitions might be unloaded and loaded in parallel as well. Both of these features will reduce significantly the elapsed time of the REORG utility. Unloading the partitions in parallel is new with DB2 9.

PIB is always activated by default and can no longer be controlled by the SORTKEYS option. There is no need anymore to specify the SORTKEYS parameter; it is the default and is ignored. Unlike the LOAD utility, SORTKEYS NO cannot be specified.

The REORG utility attempts to unload and reload partitions in parallel in some situations and does not attempt to unload and reload in parallel in other situations.

REORG attempts to unload and reload table space partitions in parallel in the following situations:

- ▶ If you specify the NOSYSREC keyword with SHRLEVEL NONE or SHRLEVEL REFERENCE.
- ▶ For SHRLEVEL CHANGE (NOSYSREC is forced).
- ▶ If you do not specify NOSYSREC with SHRLEVEL NONE or SHRLEVEL REFERENCE and specify the UNLDDN keyword with a template name, where the template's data set name pattern includes a partition number.

REORG does not attempt to unload and reload table space partitions in parallel in the following situations:

- ▶ If DATAWKnn DD statements are coded in the JCL.
- ▶ If the UTPRINT data set is not allocated to SYSOUT.
- ▶ If you do not specify the SORTDEVT keyword.
- ▶ If you specify the REBALANCE keyword.
- ▶ If rows might move from one partition to another.
- ▶ If the table space is a partition-by-growth universal table space.

If unload parallelism is possible, contiguous part ranges in REORG will all be processed by the same unload/reload subtask. To ensure that the REORG utility is able to condense the data into the minimum number of required partitions, parallelism for the REORG utility does not apply to partition-by-growth table spaces.

As with the previous utilities, the number of subtasks to achieve maximal parallelism can be constrained by available virtual memory, DB2 threads, and processors available. See 4.9, “Considerations for running parallel subtasks” on page 90 for more details.

Because with DB2 9 partitions can be unloaded and reloaded in parallel, in most cases it is no longer recommended to drop the NPIs before the REORG and rebuilding them afterwards with REBUILD INDEX. This was a possible technique to improve performance before on partitioned table spaces because REBUILD INDEX already had the capability of unloading keys in parallel. But in particular cases, this technique can still be beneficial (for example, if the NPI is very disorganized).

With DB2 9, for online REORG jobs on the partition level, the BUILD2 phase has been completely eliminated by rebuilding the entire NPIs to shadow data sets. All NPIs are now completely reorganized, even when only one partition is reorganized. Therefore, it is recommended in DB2 9 to reorganize as much contiguous partitions as possible in the same online REORG job, or even do a online REORG of the complete table space instead of using different jobs on different partitions or partition ranges. Execution of online REORG PART in parallel jobs on different partitions of the same table space is no longer allowed because of the NPI shadow data sets.

The unload/reload of the whole NPI during a REORG SHRLEVEL CHANGE (or REFERENCE) PART is essentially equivalent to a REORG INDEX of the NPI. If you currently run REORG INDEX on all NPIs following a REORG PART, this should no longer be needed and we recommend changing your existing jobs.

For more information about online REORG performance improvements in DB2 9, refer to *DB2 9 for z/OS Performance Topics*, SG24-74733.

## 4.7 Parallelism with the REBUILD INDEX utility

As with the LOAD and REORG utilities, PIB enables the REBUILD INDEX utility to build the indexes of the table space in parallel. Furthermore, if the table space is partitioned, partitions will be unloaded in parallel and the parts of partitioned indexes will be built in parallel as well. Both of these features reduce significantly the elapsed time of the REBUILD INDEX compared to old versions of DB2.

The method used by PIB is similar to the methods used by the LOAD and REORG utilities that are described in 4.6, “Parallelism with the REORG TABLESPACE utility” on page 85. It is used during both REBUILD INDEX SHRLEVEL CHANGE (on shadow data sets) or REBUILD INDEX SHRLEVEL REFERENCE (on active data sets).

PIB is always activated by default and can no longer be controlled by the SORTKEYS option. There is no need anymore to specify the SORTKEYS parameter; it is the default and is ignored.

REBUILD INDEX SHRLEVEL CHANGE jobs cannot be run to rebuild indexes on the same table space concurrently. As an alternative, REBUILD INDEX can build indexes in parallel by specifying multiple indexes in a single utility statement, using a LISTDEF or by specifying INDEX(ALL). Concurrency for rebuilding indexes in different table space is still allowed, as is the concurrency in rebuilding different partitions of an index in a partitioned table space.

As with the previous utilities, the number of subtasks to achieve maximal parallelism can be constrained by available virtual memory, DB2 threads, and processors available. See 4.9, “Considerations for running parallel subtasks” on page 90 for more details.

We also need to distinguish between the individual rebuild of a partitioned index, the individual rebuild of a non-partitioned index (NPI), and the rebuild of all indexes of a partitioned and non-partitioned table space.

### 4.7.1 Rebuilding the partitioning index of a partitioned table space

This case applies if we only rebuild the partitioning index of a partitioned table space, or if we do a REBUILD INDEX(ALL) and the partitioned table space contains no NPIs. In this case, the partitions will be unloaded in parallel, and the individual parts of the partitioned index will be built in parallel during the SORTBLD phase. PIB is always enabled.

Figure 4-5 shows three unload subtasks piping the index keys to three sort subtasks and then onto three build subtasks. In this example, inline statistics were not gathered (otherwise there would be three more subtasks).

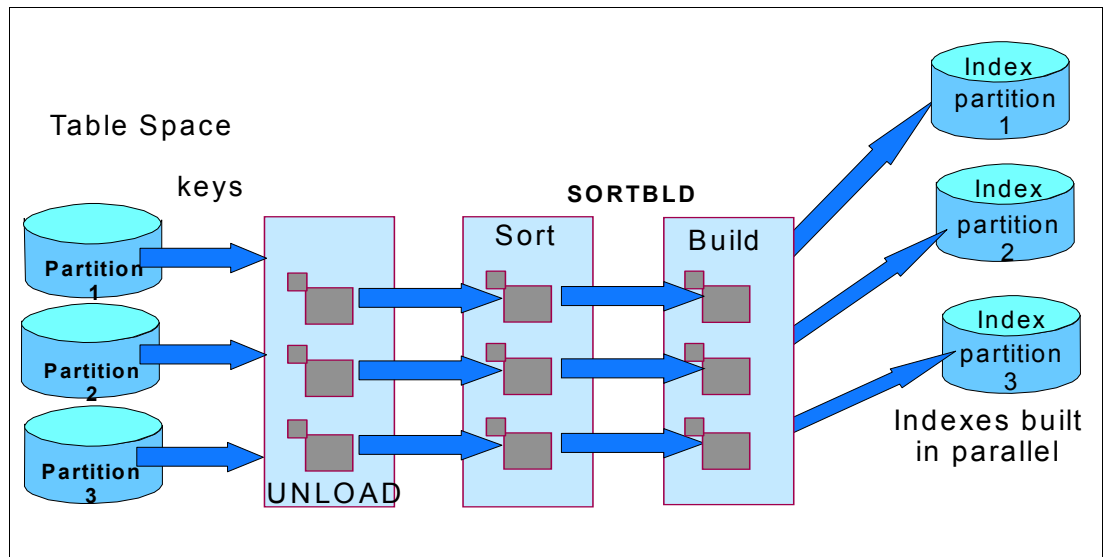


Figure 4-5 Rebuilding a partitioned index with PIB

If you use the PART option to rebuild only a single partition of an index, the utility does not need to scan the entire table space.

## 4.7.2 Rebuilding a non-partitioning index

This case applies if we rebuild a non-partitioning index of a partitioned table space. In this case, the partitions will also be unloaded in parallel, and the NPI will be built by a subsequent MERGE and BUILD task. PIB is always enabled.

When building a non-partitioning index, the unload and sort run in parallel, and the sort subtasks pipe the data to a single merge subtask that in turn pipes the data to a single build task. If inline statistics are collected, then there will be a single statistics subtask. This is shown in Figure 4-6.

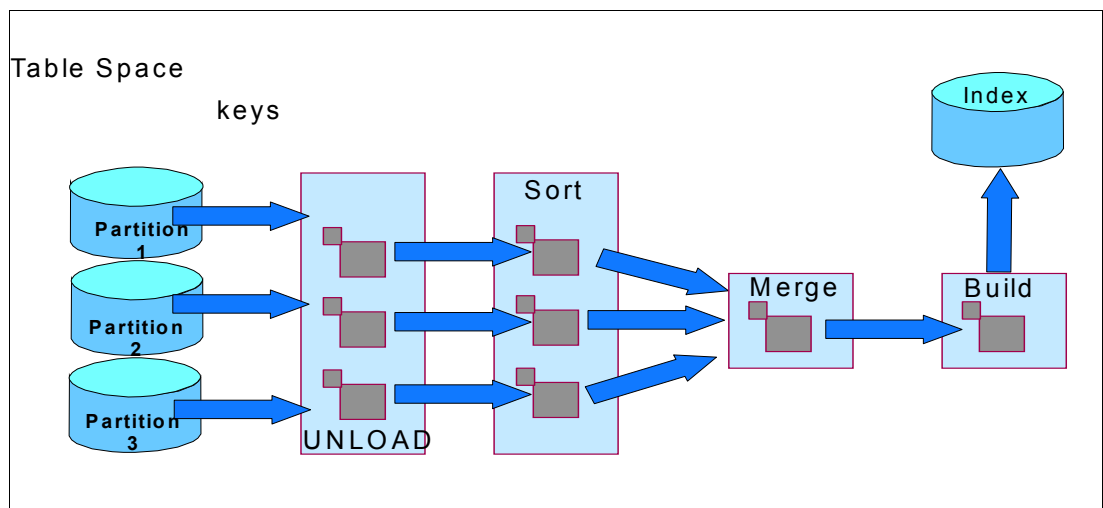


Figure 4-6 Rebuilding a non-partitioned index with PIB

With PIB, it is no longer required to submit separate REBUILD INDEX PART n jobs to unload the keys, then merge the SYSUT1 outputs in order to create a large NPI. DB2 can now easily create a large NPI using the REBUILD INDEX.

### 4.7.3 Rebuilding all indexes of a partitioned table space

This case applies if we do a REBUILD INDEX(ALL) on a partitioned table space that contains at least one NPI. In this case, the table space will also be unloaded in parallel by as many subtasks as indexes need to be built, and the indexes will be built in parallel during the SORTBLD phase. PIB is always enabled.

Unlike the case where the partitioned table space has no NPIs, the building of the partitioned index is done on the entire index level. This occurs because building the parts of the partitioned index in parallel would not decrease the total elapsed time of the utility job.

Figure 4-7 shows three unload subtasks piping the index keys to three sort subtasks and then onto three build subtasks. In this example, inline statistics were not gathered (otherwise, there would be three more subtasks).

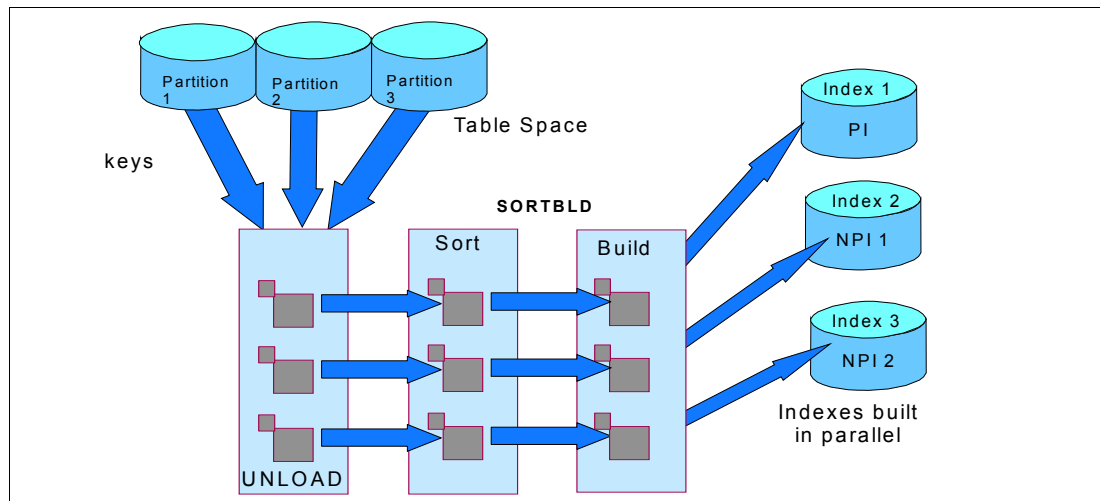


Figure 4-7 Rebuilding all indexes of a partitioned table space with PIB using SORTKEYS

You can force DB2 to build the parts of the partitioned index in parallel as well, by specifying in the REBUILD INDEX command a list of all the individual index parts with the PART keyword, or by using a list defined in a LISTDEF with the PARTLEVEL keyword. However, this approach is not recommended, as it will increase the CPU time and not reduce the elapsed time of the whole utility job.

As already mentioned, with REBUILD INDEX SHRLEVEL CHANGE, you can no longer REBUILD the indexes in different parallel jobs, because of the shadow data sets. Use one job with INDEX(ALL) instead.

### 4.7.4 Rebuilding all indexes of a non-partitioned table space

This case applies if we do a REBUILD INDEX(ALL) on a non-partitioned table space that contains at least two indexes. In this case, the indexes will also be built in parallel during the SORTBLD phase and PIB is always enabled.

Figure 4-8 shows three sort subtasks and three build subtasks for building three indexes in parallel. In this example, inline statistics were not gathered (otherwise, there would be three more subtasks).

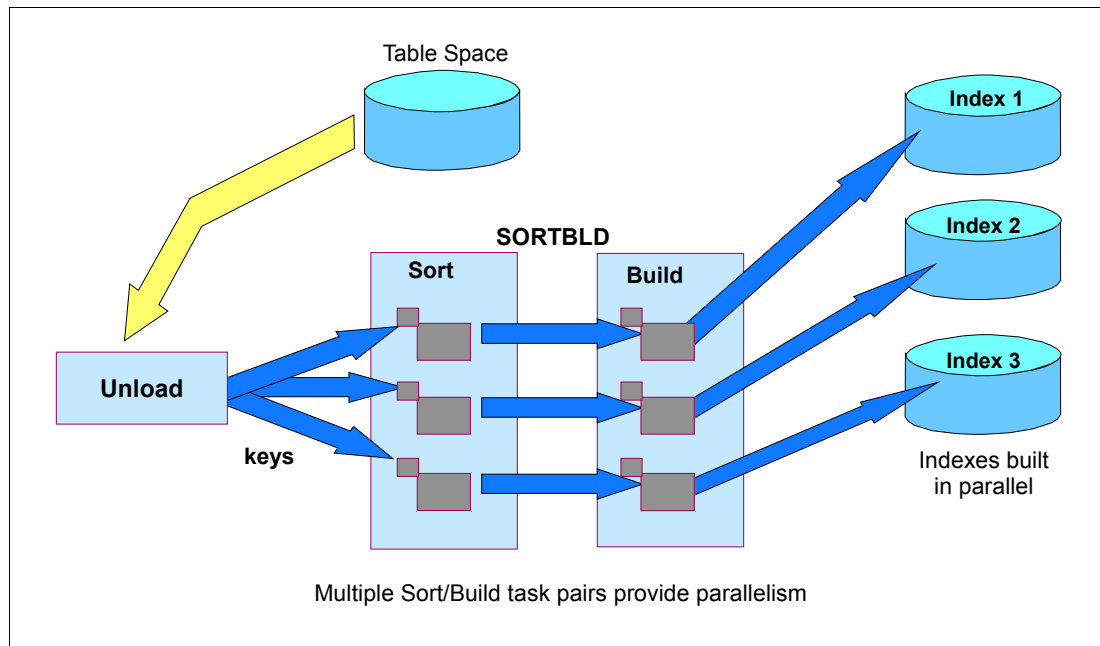


Figure 4-8 Rebuilding all indexes of a non-partitioned table space with PIB using SORTKEYS

REBUILD INDEX SHRLEVEL CHANGE it is no longer allowed to REBUILD the indexes in different parallel jobs, because of the shadow data sets. Use one job with INDEX(ALL) instead

## 4.8 Parallelism with the CHECK INDEX utility

Starting with DB2 9, if you specify more than one index to check both CHECK INDEX SHRLEVEL REFERENCE and SHRLEVEL CHANGE, check the indexes in parallel. Checking indexes in parallel reduces the elapsed time for a CHECK INDEX job by sorting the index keys and checking multiple indexes in parallel, rather than sequentially. See Chapter 6, “Sort processing” on page 141 for details about DB2 sort processing and DB2 allocated sort work data sets.

We can distinguish between check index on a non-partitioned table space, check index of a single partition of a partitioned table space, and check index on all partitions of a partitioned table space. In case of a partitioned table space, DB2 will also try to unload the partitions in parallel.

As with the previous utilities, the number of subtasks to achieve maximal parallelism can be constrained by available virtual memory, DB2 threads, and processors available. Refer to 4.9, “Considerations for running parallel subtasks” on page 90 for more details.

Refer to *DB2 Version 9.1 for z/OS Utility Guide and Reference*, SC18-9855 for examples and figures.

## 4.9 Considerations for running parallel subtasks

The actual number of parallel subtasks scheduled by DB2 in a utility job can be less than the maximum expected number. The degree of parallelism, that is, the number of parallel subtasks, will be influenced by the following factors:

- ▶ Number of page sets to be processed.
- ▶ Available virtual storage in the batch region.
- ▶ Number of processors (without zIIP or zAAP processors).
- ▶ Available number of DB2 threads and connections.
- ▶ Buffer pool sizes and thresholds.
- ▶ The UTSORTAL parameter is specified.

LOAD, REORG, and REBUILD limit the degree of parallelism at three times the number of processors on the LPAR. UNLOAD limits the degree of parallelism to one times the number of processors on the LPAR. Parallel index build (PIB) used by the LOAD, REORG, and REBUILD utilities is not limited by the number of processors, as the DFSORT memory requirements generally limit the parallelism in this regard. Each sort work data set consumes both above-the-line and below-the-line virtual storage, so if you explicitly specify a value for SORTNUM that is too high, the utility might decrease the degree of parallelism due to virtual storage constraints, and possibly decreasing the degree down to one, meaning no parallelism at all. Let DB2 decide based on real-time values (see Chapter 6, “Sort processing” on page 141.)

Each sort subtask uses its own sort work data sets and its own message data set. DB2 cannot use more subtasks than the number of data sets allocated. Manually allocating the sort work data sets or the sort message data sets is a way to control the number of subtasks. It is no longer recommended, but DB2 still propagates the hardcoded allocation of SORT message data sets to control the degree of parallelism. This is the case where the user intentionally does not want to run too many sort tasks in parallel when the utility would otherwise decide on how many parallel sort tasks there should be.

With many utilities now running parallel subtasks, the following items are also worth noting:

- ▶ DB2 no longer counts DB2 utility subtasks towards the CTHREAD and IDBACK thresholds, to prevent application threads to be impacted by utility jobs (DB2 V8 APAR PK26989 and DB2 9 base code). However, these values are still considered when deciding on the degree of parallelism, so ensure that IDBACK is large enough to accommodate all the parallel threads that are required to run concurrently. The number of threads required per utility varies from utility to utility, and also between phases within the same utility, and is dependent upon the options specified. We recommend increasing IDBACK to a number that is unlikely to be reached to avoid any failures. If this is problematic, then a scheduling product, such as IBM Tivoli® Operation Planning and Control (OPC), could also be used to control the number of jobs running concurrently, and thus the number of threads within DB2. CTHREAD may also need increasing in line with IDBACK.
- ▶ Virtual storage requirements will rise depending upon the number of subtasks being run by the utility. Ensure that the REGION parameter is large enough.
- ▶ The amount of real storage available will have to be considered. DB2 will limit the number of subtasks dependent upon available virtual storage. If REGION=0M is coded on the JOBCARD, then the subtasks can have as much virtual storage as is required up to the system specified limits (see “Region size explained” on page 377). With tens of subtasks being not uncommon, this could cause problems with real storage exhaustion (excessive paging). See 14.1.1, “Preparing for utility execution” on page 376.



- ▶ Increasing the number of parallel subtasks will also increase the CPU consumption of the utility jobs.
- ▶ As always, sufficient disk space will have to be available to support the multiple sort and work data sets.
- ▶ The higher the degree of parallelism, the less disk space is required to perform the sort of the index keys.
- ▶ Using templates for the work data sets is the best way to enable maximum parallelism.
- ▶ Use the smallest possible SORTNUM value to allow the maximum degree of parallelism and minimum sort impact, or consider using DB2 allocated sort work data sets, often referred to as SORTNUM elimination, to let DB2 determine the optimum number of sort data sets (UTSORTAL=YES). See “Deleting all the sort related DD cards (SORTWKnn, SW01WKnn, DATAWKnn, DA01WKnn, STATWKnn, and ST01WKnn,UTPRINnn,DTPRINnn) from the JCL., except UTPRINT and SORTDIAG” on page 157 for more information.
- ▶ In some cases, parallelism will not be triggered when the SORTDEVT keyword is not specified. DB2 needs SORTDEVT to trigger dynamic allocation of sort work data sets (either in DFSORT or in DB2), but that value has no influence on the actual number of parallel tasks chosen.
- ▶ The utility job output provides information about the degree of parallelism and an indication of whether parallelism was constrained. Look for the following messages:
  - DSNU364I - PARTITIONS WILL BE LOADED IN PARALLEL, NUMBER OF TASKS = nnn
  - DSNU395I - DSNU395I DSNURPIB - INDEXES WILL BE BUILT IN PARALLEL, NUMBER OF TASKS = nnn
  - DSNU397I - DSNU397I DSNUxxxx - NUMBER OF TASKS CONSTRAINED BY yyyy
  - DSNU427I DSNUBBID - OBJECTS WILL BE PROCESSED IN PARALLEL, NUMBER OF OBJECTS = nnn
- ▶ We strongly recommend monitoring the DSNU397I message for constraint information. See Example 4-3, where the number of connections is constrained because of an IDBACK value that is too small.

*Example 4-3 The monitoring DSNU397I message*

---

```

DSNU000I 273 14:12:33.37 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = DAVY00
DSNU1044I 273 14:12:33.42 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
DSNU050I 273 14:12:33.42 DSNUGUTC - REBUILD INDEX(S.I_MTSTBO_4) STATISTICS KEYCARD REPORT YES
DSNU3340I 273 14:12:33.47 DSNUCRIB - UTILITY PERFORMS DYNAMIC ALLOCATION OF SORT DISK SPACE
DSNU395I 273 14:12:34.18 DSNUCRIB - INDEXES WILL BE BUILT IN PARALLEL, NUMBER OF TASKS = 36
DSNU397I 273 14:12:34.18 DSNUCRIB - NUMBER OF TASKS CONSTRAINED BY CONNECTION
DSNU3342I 273 14:12:34.18 DSNUCRIB - NUMBER OF OPTIMAL SORT TASKS = 25, NUMBER OF ACTIVE SORT TASKS = 9

```

---





## Invoking and controlling executions

Every DB2 environment has one method or another to execute their utilities on DB2 objects during their “housekeeping” window. Various triggering limits are often used, either from within the utility control statements, or external to the utilities. Traditionally, either ISV products or in-house written programs are used to query the DB2 catalog and generate the utility statements. Starting with DB2 V7, DB2 provided stored procedures can also be used to query real-time statistics (RTS) values, which provide more accurate information and are collected anyway and stored in the DB2 catalog as of DB2 9.

A few DB2 environments merely run utilities on a regular basis; they may take an image copy nightly or use some scheduled process to drive the utility execution. However, precious resources might be wasted by executing these utilities on objects that, for example, do not need to be image copied because they have little or few updates or perhaps their applications use index-only access to the data and may not benefit from being reorganized weekly.

In this chapter, we provide some guidance as to what factors to consider before taking that image copy, running that REORG, or invoking the RUNSTATS utility. All too often, the criteria to run one of these utilities are ignored. This guidance is intended to allow you to save on using unnecessary resources. However, the guidelines are merely that: They are only general recommendations without knowledge of the application and the miscellaneous and possibly impelling factors in your installation. Only you understand the application and the application enforced referential integrity, know your recoverability requirements, and understand the other environmental factors than go into making such a decision.

In this chapter, we cover these topics:

- ▶ Triggering utility executions
- ▶ Important DB2 catalog space statistics
- ▶ RUNSTATS
- ▶ What is new with real-time statistics (RTS)
- ▶ Trends from historical statistics
- ▶ Stored procedure DSNACCOX
- ▶ DB2 Administrative Task Scheduler
- ▶ IBM DB2 Tools

## 5.1 Triggering utility executions

Triggering utilities on DB2 objects only when necessary is the preferred methodology of most DB2 environments. DB2 environments do not have the resources to waste and more and more DBAs are setting up standard procedures to invoke utilities. The best utility is not having to run a utility. You should only execute REORG, RUNSTATS, and COPY when they are really needed.

Usually, triggering utility execution is a two-step process:

- ▶ The first step consists of identifying some criteria to be used in determining what utility should be run; often this is based on one or more statistical values in the DB2 catalog related to the DB2 object. These values were traditionally periodically collected via RUNSTATS, but the trend is towards using the “real-time” values of RTS.
- ▶ The second step consists of utilizing these statistical values, directly or indirectly, to decide on the activation of the relevant utility function on the DB2 object.

We describe the catalog space statistical fields within the DB2 catalog in detail and illustrate how these fields can be used to trigger different utility execution. We also cover the history catalog tables and their use in calculating trends in growth in DB2 objects, such as determining the compression and decompression dictionary build.

Another important relevant topic is RTS, especially now that it is part of the DB2 catalog and implicitly run. Use RTS for space statistics rather than executing RUNSTATS to determine when to run other utilities. A huge advantage to using RTS is that it has no cost to you at all; RTS are automatically generated and maintained in DB2.

Two sections deal with stored procedures and the IBM DB2 tools. In these sections, we show examples of how the stored procedure can be used to control the utility functions on DB2 objects. In addition, we have a high level overview of the IBM DB2 tools that can help from a utility perspective. A more detailed description of these tools can be found in Appendix A, “DB2 Tools” on page 413.

Why and when is RUNSTATS needed? That is, what statistics should be gathered, and when do you really need to gather them? If you are using RTS for triggering utilities because they are more accurate in time and values, the most important reason to collect RUNSTATS is for optimizer statistics and special distributions like distribution statistics (DSTATS), or specific column grouping. In this case, BINDs and PREPAREs may need to be synchronized to pick up the new statistics. With static SQL, BINDs and RUNSTATS could be scheduled to run at different frequencies than in the case of dynamic SQL.

REORG has been traditionally executed for space reclamation and to improve access performance. Starting with V8, it has become necessary to consolidate structure changes implemented via online schema changes. If the reason is performance, then one needs to consider if moving rows in clustering order will help the application. It may or may not depending on the chosen access path within the application. Knowing more about the application helps you to make the best decision.

How frequently to run a COPY depends on a number of factors. The frequency of COPY is related to the window of recoverability and the expected performance during recovery. In order to recover a set of objects quickly, it is important to define correctly the frequency of image copy.

## 5.2 Important DB2 catalog space statistics

In this section, we discuss some of the more important DB2 catalog space statistics that can be used to trigger different utilities. We focus on three utilities:

- ▶ COPY
- ▶ REORG INDEX
- ▶ REORG TABLESPACE

Several indicators can be used to trigger a reorganization for better query performance or an image copy for recoverability. The following indicators are discussed in this section:

- ▶ CLUSTERRATIOF and DATAREPEATFACTOR
- ▶ NEAROFFPOSF, FAROFFPOSF, and CARDF
- ▶ NEARINDREF and FARINDREF
- ▶ Relationship of \*OFFPOS, \*INDREF, and CLUSTERRATIOF
- ▶ CHANGELIMIT option
- ▶ AVGWLEN and AVGKEYLEN
- ▶ LEAFNEAR and LEAFFAR
- ▶ LEAFDIST and LEAFDISTLIMIT

### 5.2.1 CLUSTERRATIOF and DATAREPEATFACTOR

SYSIBM.SYSINDEXES contains the column for CLUSTERRATIOF. It represents a distribution statistic that helps access path selection as well as in determining when to reorganize a table space. CLUSTERRATIOF represents the percentage of rows that are in clustering order. Rows are counted as being clustered if they are in a greater than or equal page number of the previous row. A value of 1 indicates all rows are in clustering order, whereas a value of .7 indicates 70% of the rows are in clustering order.

This is a statistic that describes the data in the table space, even though it is reported in SYSIBM.SYSINDEXES and SYSIBM.SYSINDEXES\_HIST.

Normally, a REORG TABLESPACE resets the CLUSTERRATIOF back to 1 (which represents 100%) and indicates that the table space is in clustering order. REORG INDEX does not affect this statistic.

There is a new DSNZPARM STATCLUS that represents the STATISTICS CLUSTERING field. Its acceptable values are ENHANCED or STANDARD, and it specifies the type of clustering statistics that are to be collected by the RUNSTATS utility. The default is ENHANCED. Refer to Table 14-8 on page 402 for additional information about this DSNZPARM.

Table 5-1 illustrates the impact of a high and low cluster ratio with a high and low data repeat factor.

*Table 5-1 CLUSTERRATIOF and DATAREPEATFACTOR impact*

CLUSTERRATIOF mode	High DATAREPEATFACTOR	Low DATAREPEATFACTOR
High CLUSTERRATIOF	Sequential but not dense	Density not matching clustering
Low CLUSTERRATIOF	Random index	Small table

The problems with the older STANDARD calculation are summarized in Table 5-2. Depending on the density of the clustering key values, it sometimes resulted in the DB2 optimizer choosing an access path that may not have been optimal when the data was not densely clustered.

The STANDARD calculation could result in:

- Higher values on those indexes with:
  - Many duplicates (lower COLCARDF) in recognition of sequential RIDs
  - Smaller tables due to less cluster ratio degradation from random inserts
  - Indexes that are reverse sequential
  - Lower values for random indexes with no benefit from dynamic prefetch

*Table 5-2 Limitations of the STATCLUS=STANDARD calculation*

STATCLUS=STANDARD limitation	Problem
Counts a row as clustered if the next key resides on an equal or forward page of the last RID of the current key.	The next key value could be 1000 pages away. It is difficult to determine if the gap to the next key could benefit from prefetch, based on the current prefetch quantity. This results in an overestimated cluster ratio and non-optimal access paths, an inappropriate usage of sequential or list prefetch, and the DB2 optimizer deciding not to use an index based upon a perceived high cluster ratio.
The methodology does not consider a reverse clustering sequence.	Dynamic prefetch can be activated on trigger values in forward or reverse sequence. The STANDARD calculation only considers sequential prefetch that supports forward access.
Lack of enough information about data density.	The calculation does not take into account the density of the keys. It is important to know if retrieving a certain number of rows via a clustered index results in a few data pages (dense data) or if they are scattered across many pages (non-dense data).
Only distinct key values (not the RIDs) are counted.	Non unique key values were not examined. Lower cardinality indexes may be at a disadvantage and not chosen due to the expectation of a high percentage of random I/Os, whereas sequential RID chains can benefit from dynamic prefetch. Prefetch may not be the best choice. This can also lead to unnecessary repetitive index scans.

The ENHANCED calculation tries to solve each of these problems:

- It tracks pages within a sliding prefetch window that considers both the prefetch quantity and buffer pool size. The page is considered clustered only if it falls within this window.
- Dynamic prefetch is used more widely for regular index access. Pages are counted as clustered in either direction, because the direction can change from forward to reverse and still be considered clustered.
- A new SYSINDEXES catalog table column, DATAREPEATFACTOR, tracks whether rows are found on a page other than the current. Used in conjunction with the cluster ratio value, the optimizer can now differentiate between dense and non-dense and make better choices especially regarding prefetch.
- Each RID is counted when there are multiple rows per index key value.
- Can differentiate between dense and sequential versus sequential (not dense).

When using the ENHANCED calculation, you will see some differences in the cluster ratio values in the following cases:

- ▶ Indexes with many duplicate key values
- ▶ Tables that are clustered, but in a non-dense fashion

In order to get the new ENHANCED cluster ratio values taken into account, you need to run RUNSTATS and then REBIND your static programs.

The ENHANCED clustering statistics result in an improved cluster ratio formula and uses the new statistic DATAREPEATFACTOR. These statistics allow for improved SQL access paths based on improved recognition of the number of getpages required to retrieve the qualified rows and whether those getpages benefit from prefetch. In only rare cases where you might experience a problem, you might consider reverting back to the STANDARD calculation. We recommend using the default of ENHANCED.

### 5.2.2 NEAROFFPOSF, FAROFFPOSF, and CARDF

NEAROFFPOSF, FAROFFPOSF, and CARDF in SYSINDEXPART provide information about the disorganized status of a table space with clustering indexes. These columns are updated by RUNSTATS with the UPDATE ALL or UPDATE SPACE options.

- ▶ *CARDF* is the number of rows in the table space.
- ▶ *NEAROFFPOSF* is the number of times it would be necessary to access a different, “near-off” page when accessing all the data records in index order.
  - For non-segmented table spaces, a page is considered near-off the present page if the difference between the two page numbers is greater than or equal to 2, and less than 16.
  - For segmented table spaces, a page is considered near-off the present page if the difference between the two page numbers is greater than or equal to 2, and less than  $SEGSIZE * 2$ .
  - A large value for NEAROFFPOSF might indicate that table space clustering is deteriorating. However, NEAROFFPOSF is not as critical a factor as FAROFFPOSF.
- ▶ *FAROFFPOSF* is the number of times it would be necessary to access a different, “far-off” page when accessing all the data records in index order.
  - For non-segmented table spaces, a page is considered far-off the present page if the two page numbers differ by 16 or more.
  - For segmented table spaces, a page is considered far-off the present page if the two page numbers differ by  $SEGSIZE * 2$  or more.
  - A large value for FAROFFPOSF might indicate that table space clustering is deteriorating and is more critical than NEAROFFPOSF. It is more significant, because a synchronous I/O is issued when the next page is not within the prefetch range.

**Important:** Do not use FAROFFPOSF and NEAROFFPOSF to determine whether to reorganize DB2 catalog table spaces SYSDBASE, SYSDBAUT, SYSGROUP, SYSVIEWS, and SYSPLAN.

## OFFPOSLIMIT option

OFFPOSLIMIT is an indicator of the degradation of the clustering index of a table within the table space. When rows are accessed by clustering index, performance can be affected due to high synchronous I/O for those data pages not within the prefetch range. When the CLUSTERRATIOF is 100%, this is an indication that data in the table space is in clustering order. In Figure 5-1, we have a table space on the left that is in non-clustering order. After a REORG, you see the same table space on the right in clustering order.

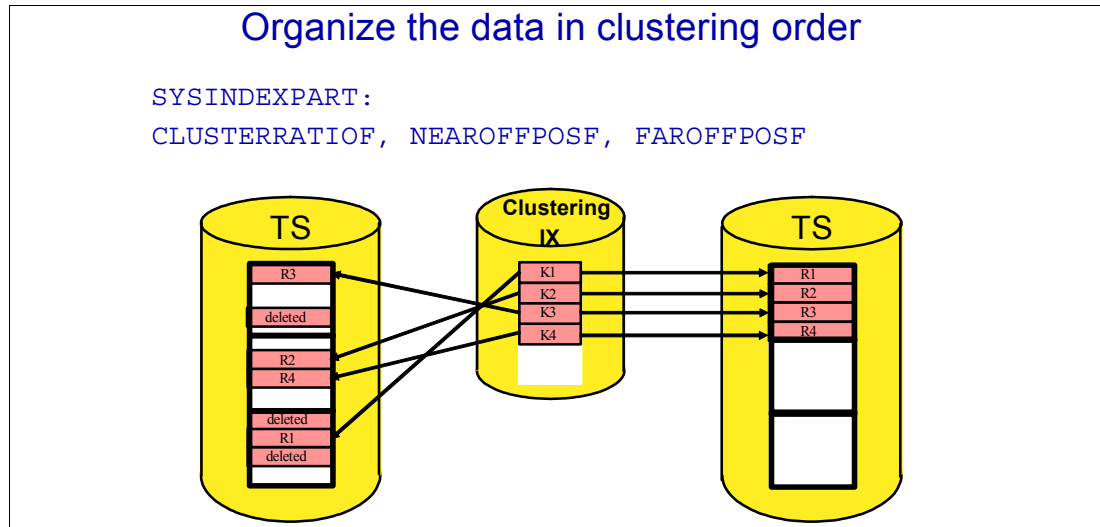


Figure 5-1 Non-clustering and clustering order

NEAROFFPOSF, FAROFFPOSF, and CARDF are used in the computation of the option OFFPOSLIMIT to trigger a REORG TABLESPACE:

$$\text{OFFPOSLIMIT} = ((\text{NEAROFFPOS} + \text{FAROFFPOS}) / \text{CARDF}) * 100$$

A REORG TABLESPACE with the OFFPOSLIMIT option is triggered if the OFFPOSLIMIT is exceeded. The default value for OFFPOSLIMIT is 10.



### 5.2.3 NEARINDREF and FARINDREF

When an update to a row is performed, the resulting length of the row may be longer than its original length. If the updated row cannot fit into its original page, DB2 places it in another page. However, the original RID in the index still points to the former page. When this occurs, an entry is made in the original data page that is a pointer to the new page where the updated row is stored. This is called an indirect row reference or an overflow record. The diagram in Figure 5-2 is an example of such an indirect reference. There usually is additional I/O involved to read the extra page into a buffer pool.

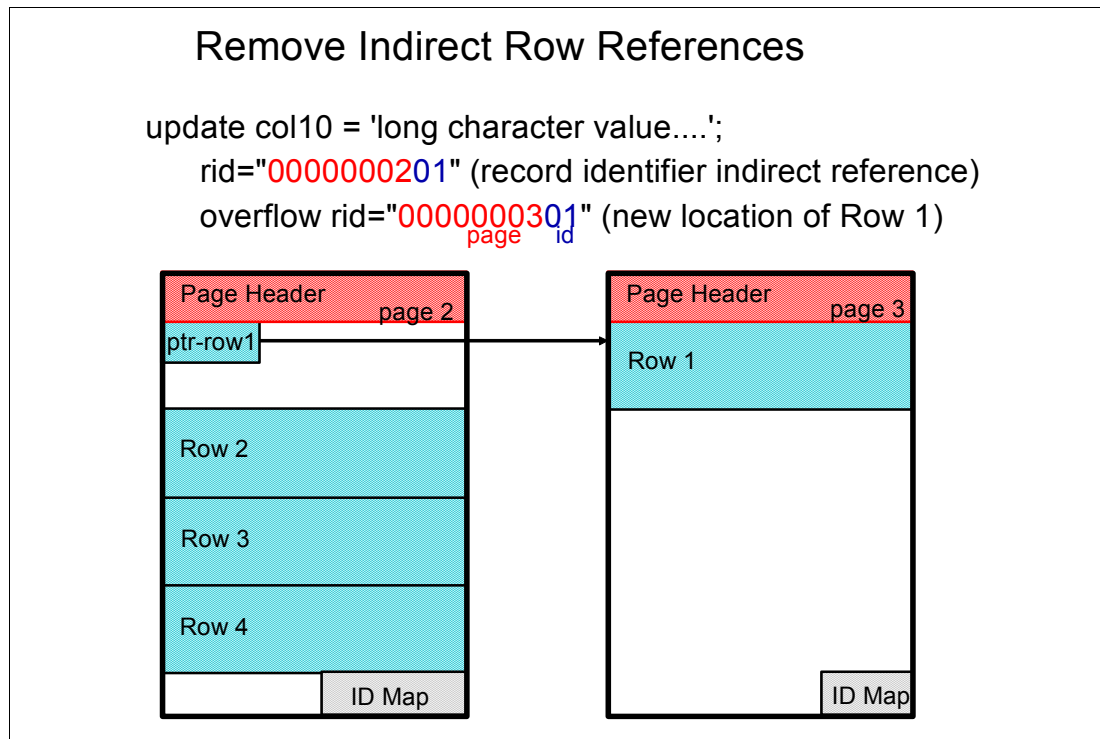


Figure 5-2 Remove indirect row reference

Two values in SYSTABLEPART provide quantitative values for the measurement of indirect addressing:

- ▶ NEARINDREF
- ▶ FARINDREF

These values are updated by RUNSTATS with the UPDATE ALL or UPDATE SPACE options:

- ▶ NEARINDREF is the number of indirect references to overflow rows on another page within 64 pages.
- ▶ FARINDREF is the number of indirect references to overflow rows outside the 64 page limit.

Both NEARINDREF and FARINDREF indicate disorganization of the data. When rows are placed in another page other than their home page, lock avoidance techniques cannot be used, so DB2 has to take a lock. That can lead to performance degradation, especially in a data sharing environment.

## INDREFLIMIT

INDREFLIMIT is another indicator of the degradation of the table within the table space. Too many overflow records can severely affect query performance characteristics; you may see more pages being read synchronously than necessary.

The REORG utility calculates the INDREFLIMIT value as:

$$\text{INDREFLIMIT} = ((\text{NEARINDREF} + \text{FARINDREF})/\text{CARDF}) * 100$$

If this value exceeds the specified value in REORG, then REORG is performed. The default value for INDREFLIMIT is 10.

We recommend that you run REORG TABLESPACE if the percent of indirect row references is greater than 10% for a non-data sharing environment, and 5% for a data sharing environment. You may also want to consider changing the PCTFREE value to allow more expansion in the table space in an effort to try to minimize the number of overflow records needed.

**Note:** If the partitioned table space is reorganized without the PART keyword, then the limits are calculated for each partition. REORG is triggered when the limits exceed any one of the partitions.

**Note:** When a table space is reorganized, so are any associated index spaces. If you have a job stream that generates REORG statements, first you generate reorganization of table spaces, followed by index spaces. Your generator job should be able to remove reorganization of index spaces when the associated table space is reorganized.

### 5.2.4 Relationship of \*OFFPOS, \*INDREF, and CLUSTERRATIOF

How does the different statistical metrics NEAROFFPOS/FAROFFPOS and NEARINDREF/FARINDREF relate to CLUSTERRATIOF?

\*INDREF correlates closely with the cluster count if the keys are in cluster order and then rows are relocated to another page. However, cases exist where these statistics are correlated and cases exist where these statistics are not correlated.

Figure 5-3 illustrates an example where INDREF is correlated with CLUSTERRATIOF. Notice here there are four indirect references to C, D, G, and K. The optimal cluster count would be 11, but here it is only 8.

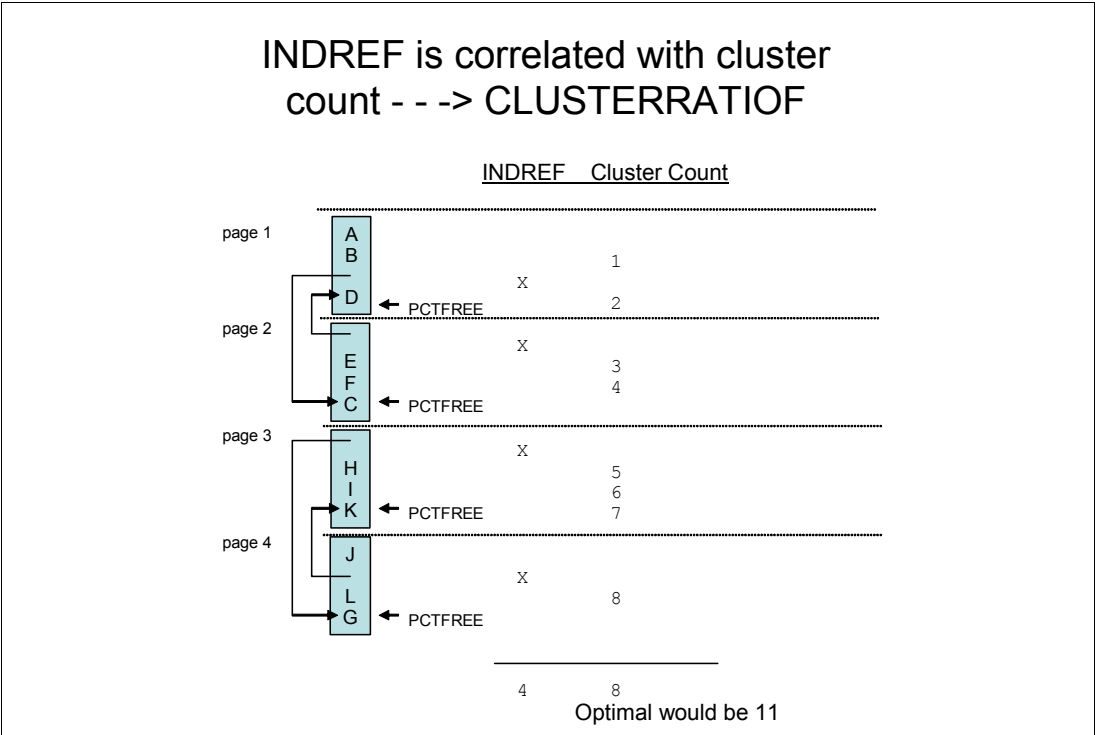


Figure 5-3 INDREF is correlated with cluster count

Refer to Figure 5-4, which illustrates an example where INDREF is *not* correlated with CLUSTERRATIOF. Notice here there are four indirect references to C, D, G, and K. The optimal cluster count is 11 and actual cluster count is also 11, so it is perfectly clustered. There is no correlation between INDREF and cluster count.

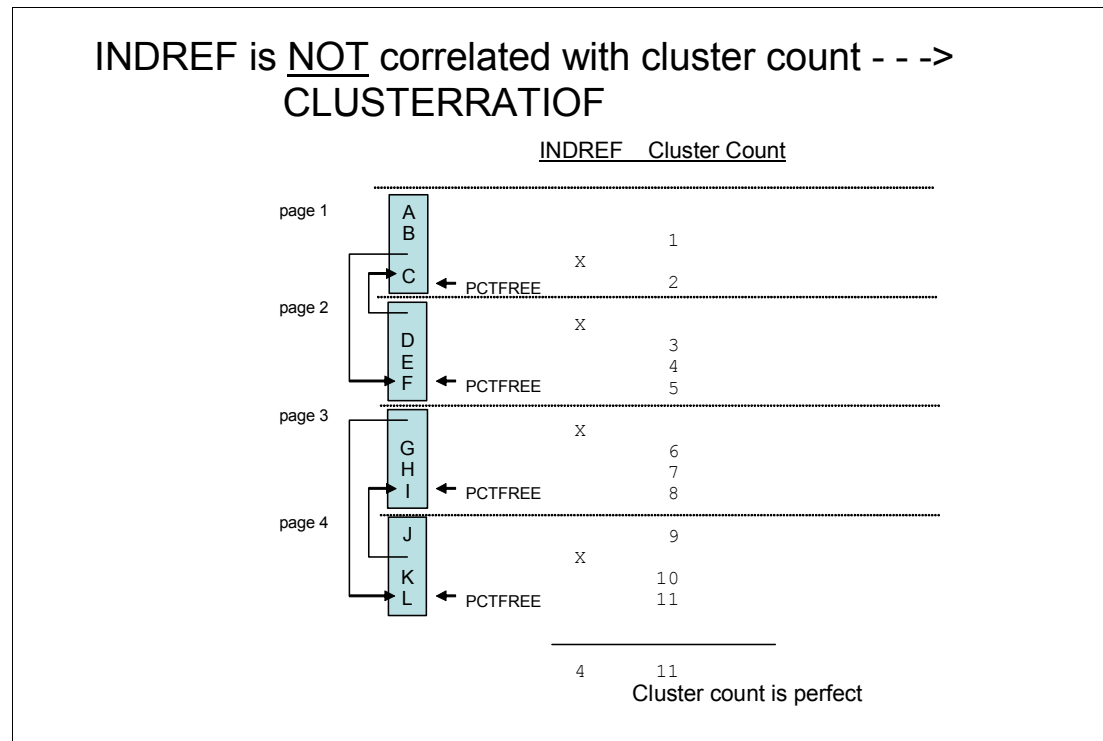


Figure 5-4 INDREF is not correlated with cluster count

\*OFFPOS directly affects the cluster count. A single “jump” counts as two OFFPOS’, so the cluster count should be half of the sum of the \*OFFPOS’. In Figure 5-5, notice the actual cluster count is 8 as compared to the optimal cluster count, which is 11. There are six index rows with key values C, D, G, H, K, and L that are counted toward \*OFFPOS. Here the cluster count is half of the sum of the \*OFFPOS, which is 6 or 3. Notice that OFFPOS 3 is correlated with a cluster count of 8, because  $3 + 8 = 11$ .

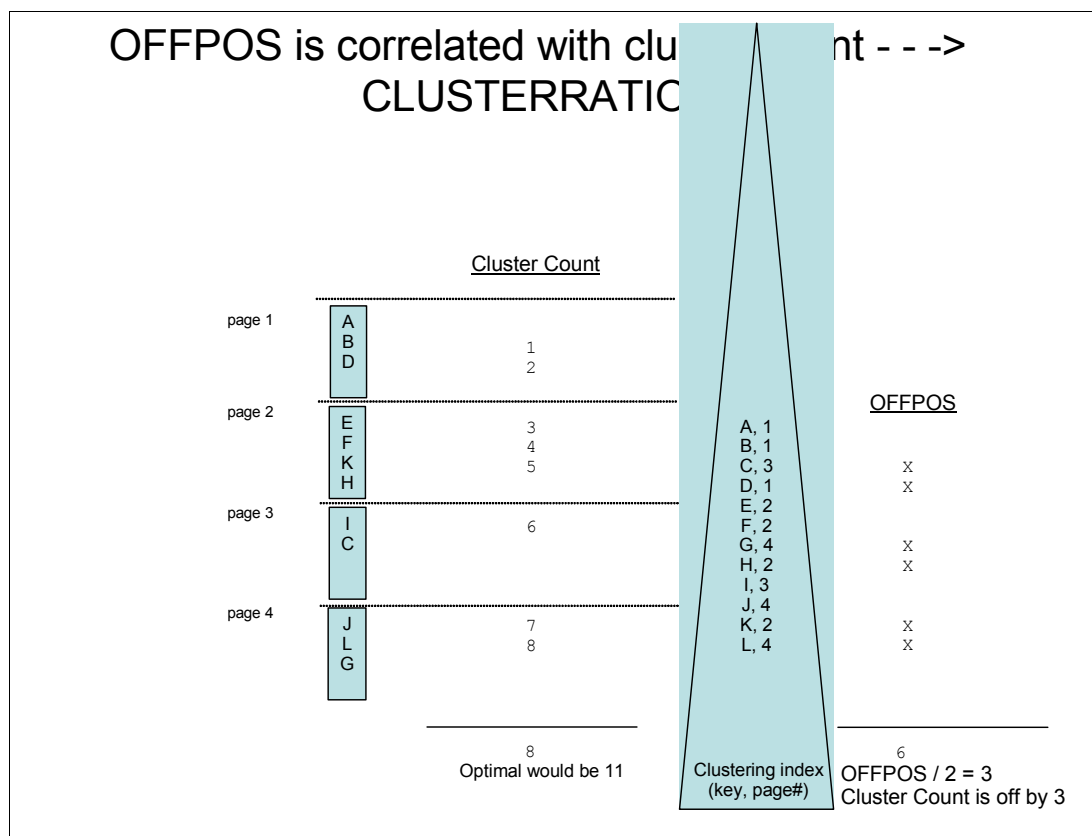


Figure 5-5 OFFPOS and CLUSTERRATIO correlation

## 5.2.5 CHANGELIMIT option

There are no statistical field values in SYSTABLEPART or SYSINDEXPART that can be used to trigger the COPY utility. The only option is CHANGELIMIT, which can be used to trigger an image copy of a table space. It is calculated as follows:

CHANGELIMIT (percentage-value1,percentage-value 2)

COPY scans the header page of the table space for space map bits and the image copy bit LRSN. The result depends on whether you use a single value (pv1) or a double set of values (pv1,pv2).

Note that you can use statistics collected by RTS that include the last time an image copy was taken and the number of rows inserted, updated, or deleted since the last COPY was executed. Refer to 5.4, “What is new with real-time statistics (RTS)” on page 117 for more information.

## Single value for CHANGELIMIT

If only a single percentage-value1 (pv1) is specified for the CHANGELIMIT, then an image copy for the table space is triggered based on the criteria listed in Table 5-3.

Table 5-3 COPY CHANGELIMIT with a single value

%Changed pages	COPY TABLESPACE
pv1 > 0 and %changed pages < pv1	Yes, incremental image copy
%changed pages > or = pv1	Yes, full image copy
%changed pages = 0	No image copy
CHANGELIMIT(0)	Yes, full image copy

## Double values for CHANGELIMIT

If both the lower bound and upper bound percentage values are specified (pv1,pv2), then an image copy for the table space is triggered based on the criteria listed inTable 5-4.

Table 5-4 COPY CHANGELIMIT with double values

%Changed pages	COPY TABLESPACE
pv1 < %changed pages < pv2	Yes, incremental image copy
%changed pages > or = pv2	Yes, full image copy
pv1 = pv2 AND %changed pages > or = pv1	Yes, full image copy
%changed pages < or = pv1	No image copy
CHANGELIMIT(0)	Yes, full image copy

If CHANGELIMIT(0), then a full image copy is always taken. Obviously, the additional option of REPORTONLY can be used with CHANGELIMIT, and even then, only image copy information is displayed and image copies are not taken, only recommended.

In DB2 9, the option CHANGELIMIT(ANY) was added via APAR PK42573 and PK42768. CHANGELIMIT(ANY) creates a full image copy if any page has been changed since the last image copy.

The default values for CHANGELIMIT, when specified, are (1,10).

COPY scans the spacemap pages of the table space for modified page indicators. This will recall table spaces that are migrated even with the REPORTONLY option.

**Note:** CHANGELIMIT option is not available for COPY INDEXSPACE. Furthermore, COPY does not support incremental image copy for index space.

## 5.2.6 AVGWLEN and AVGKEYLEN

In this section, we explore the usefulness of AVGWLEN and AVGKEYLEN space statistics.

## AVGROWLEN

AVGROWLEN is a space statistic that is collected at the table space and table level and at the partition level. AVGROWLEN is the average length of rows for the tables in the table space. If the table space (or partition) is compressed, then AVGROWLEN contains the average row length of the compressed data rows. It is a column in SYSTABLESPACE, SYSTABLES, SYSTABLEPART, SYSTABLES\_HIST, and SYSTABLEPART\_HIST.

It is a useful metric for estimating the current number of rows of a table space from file size without having to run RUNSTATS. In addition, you can calculate table space size allocation more accurately before using UNLOAD, REORG, and LOAD utilities or allocating work data sets when running any utility.

## Calculating table space size with AVGROWLEN

Let us look at an example of how AVGROWLEN can lead to a more accurate estimate of the space you need to allow for a table space.

Assume you have the following specifications for a moderate size table space:

- ▶ Number of records = 100000
- ▶ Maximum record size = 130 bytes
- ▶ Average record size = 80 bytes
- ▶ Page size = 4 KB
- ▶ PCTFREE = 5
- ▶ FREEPAGE = 20
- ▶ MAXROWS = 255

Using the maximum row size, you get the following results:

- ▶ Usable page size =  $4074 \times 0.95 = 3870$  bytes
- ▶ Records per page =  $\text{MIN}(\text{MAXROWS}, \text{FLOOR}(3870 / 130)) = 29$
- ▶ Pages used =  $2 + \text{CEILING}(100000 / 29) = 3451$
- ▶ Total pages =  $\text{FLOOR}(3451 \times 21 / 20) = 3624$
- ▶ Estimated number of kilobytes =  $3624 \times 4 = 14496$  KB

Using the AVGROWLEN instead of the maximum row size yields a different result:

- ▶ Usable page size =  $4074 \times 0.95 = 3870$  bytes
- ▶ Records per page =  $\text{MIN}(\text{MAXROWS}, \text{FLOOR}(3870 / 80)) = 48$
- ▶ Pages used =  $2 + \text{CEILING}(100000 / 48) = 2085$
- ▶ Total pages =  $\text{FLOOR}(2085 \times 21 / 20) = 2189$
- ▶ Estimated number of kilobytes =  $2189 \times 4 = 8756$  KB

Without knowing the average row length in a table, a cautious DBA might use the worst case scenario where all the rows are at the maximum length. From this example, we see this calculation results in an estimate of 14496 KB. However, using the AVGROWLEN value collected by RUNSTATS, we see our calculation results in a much different estimate of 8756 KB. Using the first calculation, 5740 KB of space would have been wasted. So you save 5740 KB by using AVGROWLEN instead.

## AVGKEYLEN

AVGKEYLEN is a space statistic that is collected at the index level and at the partition level. AVGKEYLEN is the average length of keys within the index. It is a column in SYSINDEXES, SYSINDEXES\_HIST, SYSINDEXPART, and SYSINDEXPART\_HIST. It is useful for estimating the current cardinality of an index space without having to run RUNSTATS. With the introduction of non-padded indexes, it is a useful metric and allows you to calculate index space size allocation more accurately.

## Calculating index space size with AVGKEYLEN

Let us look at an example of how AVGKEYLEN can lead to a more accurate estimate of the space you need to allow for a index space. For example:

- ▶ Unique Index NOT PADDED
- ▶ Number of records in table = 100000
- ▶ Key is a single column defined as VARCHAR(100) NOT NULL
- ▶ Max Key Len = 102
- ▶ AVGKEYLEN = 62
- ▶ PCTFREE = 5
- ▶ FREEPAGE = 4

If you calculate the total number of leaf pages using the maximum key size:

- ▶ Space per key =  $102 + 7 = 109$
- ▶ Usable space per page =  $\text{FLOOR}((100 - 5) \times 4038/100) = 3844$
- ▶ Entries per page =  $\text{FLOOR}(3844 / 109) = 35$
- ▶ Total leaf pages =  $\text{CEILING}(100000 / 35) = 2858$

Now calculate the total number of non-leaf pages using the maximum key size:

- ▶ Space per key =  $102 + 7 = 109$
- ▶ Usable space per page =  $\text{FLOOR}(\text{MAX}(90, (100 - 5)) \times (4046/100)) = 3836$
- ▶ Entries per page =  $\text{FLOOR}(3836 / 109) = 35$
- ▶ Minimum child pages =  $\text{MAX}(2, (35 + 1)) = 36$
- ▶ Level 2 pages =  $\text{CEILING}(2858 / 36) = 80$
- ▶ Level 3 pages =  $\text{CEILING}(80 / 36) = 3$
- ▶ Level 4 pages =  $\text{CEILING}(3 / 36) = 1$
- ▶ Total non-leaf pages =  $(80 + 3 + 1) = 84$

Now we calculate the total space required using the maximum key size:

- ▶ Free pages =  $\text{FLOOR}(2585 / 4) = 646$
- ▶ Tree pages =  $\text{MAX}(2, (2585 + 84)) = 2669$
- ▶ Space map pages =  $\text{CEILING}((2669 + 646)/8131) = 1$
- ▶ Total index pages =  $\text{MAX}(4, (1 + 2669 + 646 + 1)) = 3317$
- ▶ TOTAL SPACE REQUIRED, in KB =  $4 \times (3317 + 2) = 13276$  KB

Finally, we calculate the total leaf pages using AVGKEYLEN:

- ▶ Space per key =  $102 + 7 = 69$
- ▶ Usable space per page =  $\text{FLOOR}((100 - 5) \times 4038/100) = 3844$
- ▶ Entries per page =  $\text{FLOOR}(3844 / 69) = 55$
- ▶ Total leaf pages =  $\text{CEILING}(100000 / 55) = 1819$
- ▶ Calculate total non-leaf pages using AVGKEYLEN
- ▶ Space per key =  $62 + 7 = 69$
- ▶ Usable space per page =  $\text{FLOOR}(\text{MAX}(90, (100 - 5)) \times (4046/100)) = 3836$
- ▶ Entries per page =  $\text{FLOOR}(3836 / 69) = 55$
- ▶ Minimum child pages =  $\text{MAX}(2, (55 + 1)) = 56$
- ▶ Level 2 pages =  $\text{CEILING}(2858 / 56) = 56$
- ▶ Level 3 pages =  $\text{CEILING}(80 / 56) = 1$
- ▶ Total non-leaf pages =  $(56 + 1) = 57$



And we calculate the total space required using AVGKEYLEN:

- ▶ Free pages =  $\text{FLOOR}(1819 / 4) = 454$
- ▶ Tree pages =  $\text{MAX}(2, (1819 + 57)) = 1876$
- ▶ Space map pages =  $\text{CEILING}((1819 + 57)/8131) = 1$
- ▶ Total index pages =  $\text{MAX}(4, (1 + 1819 + 57 + 1)) = 2332$
- ▶ TOTAL SPACE REQUIRED, in KB =  $4 \times (2332 + 2) = 9336$  KB

The result: Using AVGKEYLEN only requires three levels instead of four and produces a savings of 3940 KB.

For compressed table spaces, the PAGESAVE value gives an idea of how many pages could be saved by using compression. The compression ratio also comes into play when estimating work files, as the number of records in the table space can only be determined from the number of pages when you know both AVGROWLEN and the compression ratio.

## 5.2.7 LEAFNEAR and LEAFFAR

LEAFDIST is one measurement of index leaf disorganization, but it is not as good as LEAFNEAR and LEAFFAR. LEAFDIST measures the average gap between leaf pages. However, in a large index that grows and frequently requires page splits, LEAFDIST tends to exaggerate the disorganization. This can result in too frequent and unnecessary reorganizations.

LEAFNEAR and LEAFFAR are updated by RUNSTATS with the UPDATE ALL or UPDATE SPACE options. LEAFNEAR is the number of leaf pages that are located physically near previous leaf pages for successive active leaf pages, and LEAFFAR is the number of leaf pages that are located physically far away from previous leaf pages for successive active leaf pages that are accessed in an index scan.

Essentially, LEAFNEAR and LEAFFAR more accurately measure the disorganization of physical leaf pages. This disorganization really represents the number of pages that are not in an optimal position due to index pages being deleted or leaf page splits caused by an insert that cannot fit onto a full page.

Referring to Example 5-1 on page 110, the index has almost 10,000 pages, and only four of the pages are out of position (LEAFFAR=4) with jumps. However, calculating the average gap gives a LEAFDIST value of 399, or an average jump of 3.99 pages. The value is almost twice the recommended 200 threshold for determining if the index needs to be reorganized! However, with only four leaves being out of position, reorganization is really not necessary.

LEAFFAR and LEAFNEAR were introduced to better measure physical leaf disorganization, or the number of physical leaf pages not in an optimal position. As shown in Figure 5-6, there is a logical and physical view of an index. The logical view is the tree shown at the top (two levels in our example). If we were accessing the data by scanning for keys “FRISKE” through “JACKSON”, the four leaf pages would be scanned as shown. This same scanning of pages, looking at physical page access, is shown at the bottom of Figure 5-6.

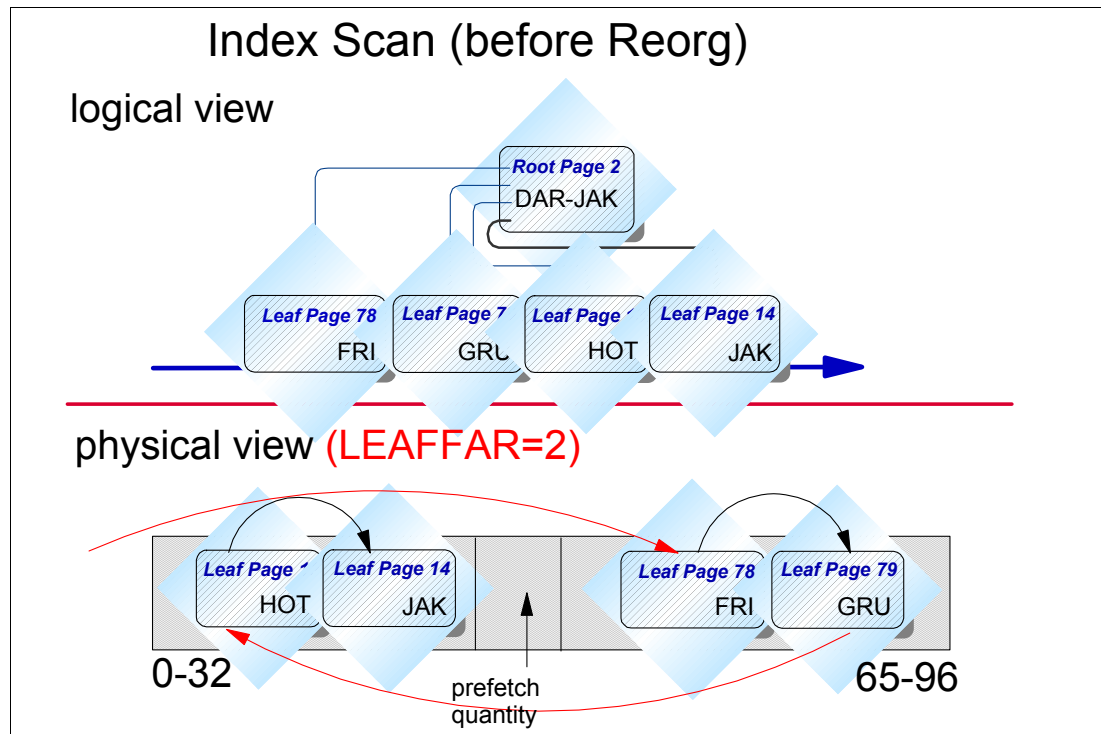


Figure 5-6 Logical and physical view before reorganization

The first page is physical page 78, and the other leaf pages are located at physical locations 79, 13, and 14. A jump backward or ahead of more than one page represents non-optimal physical ordering. As in other statistics, LEAFNEAR represents this jump within 64 pages and LEAFFAR represents the jump outside 64 pages. LEAFNEAR counts key sequential index leaf pages that are more than 1 page apart, but not adjacent.

In our example, there are two big jumps: page 1 to page 78, and page 79 to page 13. These two jumps are recorded in LEAFFAR.

When considering the statistics, the LEAFFAR value is more significant and likely to affect performance than the LEAFNEAR value, but both these values indicate disorganization of the leaf pages, which can lead to performance degradation.

# Index Scan (after Reorg)

logical view

physical view (LEAFFAR, LEAFNEAR = 0)

The percentage of leaf pages in disorganization can be calculated based on the values of LEAFFAR in SYSINDEXPART and NLEAF in SYSINDEXES as follows:

You should consider REORG INDEX if the percentage of leaf disorganization is greater than 10%. Consider changing PCTFREE or FREEPAGE to allow keys to be inserted without splitting as often.

Chapter 5. Invoking and controlling executions 109

## 5.2.8 LEAFDIST and LEAFDISTLIMIT

LEAFDIST is a column in SYSINDEXPART and SYSINDEXPART\_HIST. LEAFDIST is updated by RUNSTATS with the UPDATE ALL or UPDATE SPACE options. LEAFDIST is 100 times the average number of pages that are between successive leaf pages in the index. The formula to compute LEAFDIST is given in Example 5-1.

### Example 5-1 LEAFDIST formula

---

LEAFDIST = 100 \* Summation of distance between pages/# of leaf pages

---

If an index space is totally organized, then there are no gaps at all, and therefore the value of LEAFDIST is 0.

Leaf pages can have page gaps whenever indexes are deleted or updated, causing a leaf page to move or get inserted out of order. Index leaf page splits can also contribute to a disorganization of an index when an inserted entry cannot fit on a page. In this case, DB2 moves half of the index entries onto a new page, which may be far away from the original “home” page. The asymmetric splitting for index pages (other than the last index leaf page), introduced with DB2 9 to accommodate varying patterns of inserts into an index, should reduce this movement of pages.

The more disorganized an index gets, the higher the value of LEAFDIST. Also, FREEPAGE on an index affects the LEAFDIST calculation.

Figure 5-8 provides a simple illustration of the LEAFDIST calculation.

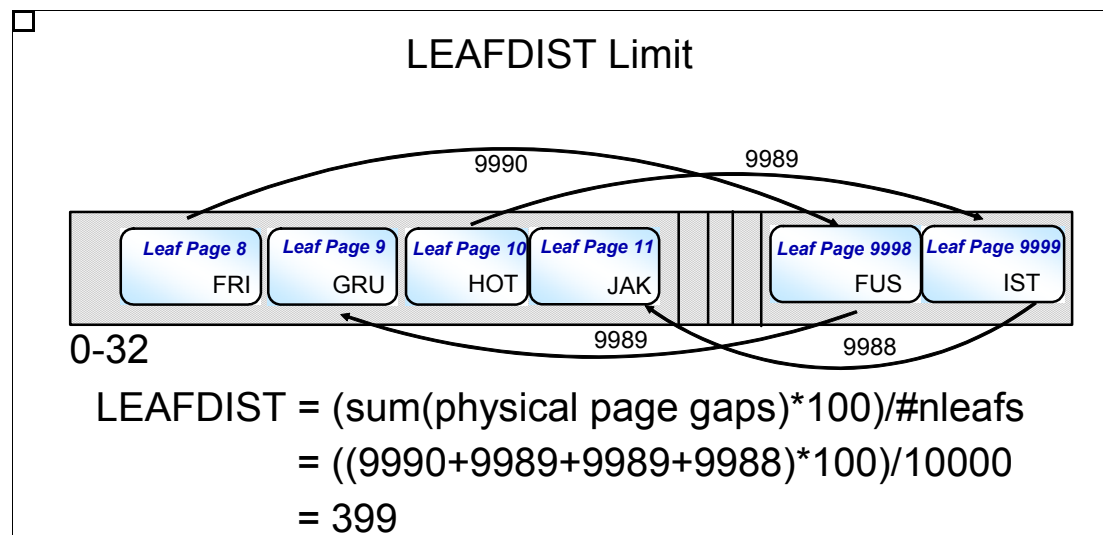


Figure 5-8 LEAFDIST calculation

In Figure 5-8:

- ▶ The navigation from the root page to leaf page 8 has no gaps.
- ▶ The navigation from leaf page 8 to leaf page 9998 has a physical page gap of 9990.
- ▶ The navigation from leaf page 9998 to leaf page 9 has a physical page gap of 9989.
- ▶ Continuing with the gap calculation, we see the resultant gaps of 9989 and 9988 respectively.
- ▶ LEAFDIST is calculated as 399.

LEAFDISTLIMIT is an option within the REORG INDEX utility where you specify a value to be used, say nnn, against each LEAFDIST in the catalog for all partitions of a specified index in SYSINDEXPART. If any LEAFDIST value exceeds the LEAFDISTLIMIT, a REORG INDEX occurs, unless you also specified the REPORTONLY option.

**Note:** The default value for LEAFDISTLIMIT is 200.

If the utility is a REORG INDEX on the PART level, then the REORG is triggered if any one of the partition index LEAFDIST values exceed LEAFDISTLIMIT. If the partition index space is reorganized without the PART keyword, then the entire index is reorganized.

If there were more gaps than active pages, LEAFDIST would be larger. We have improved statistics to determine when to run a REORG INDEX, namely LEAFFAR and some of the RTS indicators. LEAFFAR and LEAFNEAR are discussed in 5.2.7, “LEAFNEAR and LEAFFAR” on page 107, and RTS is discussed in 5.4, “What is new with real-time statistics (RTS)” on page 117.

The DB2 Administration Tool allows you to find those objects with a LEAFDIST greater than a predefined value. Refer to “Indexes with 150 or more leaf page distance” on page 424 for more information.

## 5.2.9 Trigger limits

In summary, there are REORG options that can be used to trigger the execution of the REORG utility. These limiting values are in SYSTABLEPART and SYSINDEXPART. RUNSTATS with UPDATE SPACE must be run prior to the REORG utility or the space statistics must be current. COPY table space can be triggered with CHANGELIMIT. Table 5-5 lists a summary of these KEYWORDS and the recommended limits.

*Table 5-5 Trigger limits and utilities*

Trigger KEYWORD	Limits	Remark
LEAFDISTLIMIT	> 200	REORG INDEXSPACE
OFFPOSLIMIT	>10%	REORG TABLESPACE
INDREFLIMIT	>10%	REORG TABLESPACE
CHANGELIMIT pv1	0 < %changed pages < pv1	COPY TS incremental copy
	%changed pages > or = to pv1	COPY TS full image copy
	pv1=0	
CHANGELIMIT (pv1,pv2)	pv1 < %changed pages < pv2	COPY TS incremental copy
	%changed pages > or = to pv2	COPY TS full image copy
	pv1 = pv2 AND %changed pages > or = to pv1	
	pv1=pv2=0	

The defaults for pv1 and pv2 are 1 and 10, respectively. However, according to the utility performance group, if < 5% of the pages are randomly changed, you should take an incremental image copy instead of an image copy.

## 5.2.10 VSAM data set extents

The maximum number of extents increased from 119 to 251 with DFP V1.4. Although this was a concern in older devices, it is less of a performance consideration with newer devices with a RAID architecture, where data is physically stored in logical volumes all over the I/O devices.

There are upper bounds to the number of extents. The size of each extent may influence the limit of maximum volumes per data set (59 per data set and 133 volumes per STOGROUP).

Continuous monitoring of the extents in SYSTABLEPART and SYSINDEXPART is necessary to track objects that exceed the number of extents past the threshold value of 50. When this value is exceeded, REORG should be scheduled for the object (TS or IX). You will need to review the values of PRIQTY and SEQQTY. If one or both these values have changed, then avoid the REUSE parameter, so that the data set can be deleted and recreated with the new extents. PCTFREE and FREEPAGE may also be reduced to squeeze more data into less space.

With DB2 V7, with improvements to parallel data set open, FASTSWITCH for online REORG, and the disk capabilities provided by PAV, the requirement to reduce the number of extents is certainly not so critical in terms of performance. However, it is a good practice to keep the environment under control in terms of DSMAX and VTOC size, especially from an availability perspective. You never want to run the risk of getting to a point where you have 251 extents and have to resize the object immediately and take an outage doing so.

Help is also available with tools to resize data sets and to generate an ALTER and perform a REORG, such as DB2 Administration Tool, or DB2 Storage Management Utility, and the DB2 Automation Tool. See Appendix A, “DB2 Tools” on page 413 for more information.

## 5.2.11 Reclaiming dead space

Here we have two fields that can be used to trigger REORG of table space or index space:

- ▶ PERCDROP for simple table spaces in SYSTABLEPART
- ▶ PSEUDO\_DEL\_ENTRIES for Indexes SYSINDEXPART

### Dropping tables in a simple table space

Simple table spaces can accumulate dead space that is reclaimed during reorganization. If this dead space is not regularly reclaimed, it may result in acquiring more extents than needed to add additional data rows.

For simple table spaces, dropping a table results in that table's data rows remaining. The amount of dead space for a simple table space can be tracked directly with PERCDROP in SYSTABLEPART.

We recommend that you run REORG TABLESPACE if the value of PERCDROP in SYSTABLEPART is greater than 10%.

### Removing pseudo-deleted entries

Indexes can accumulate dead space that is reclaimed during reorganization. If this dead space is not regularly reclaimed, it may result in acquiring more extents than needed to add additional keys.

For an index, deleted keys are marked as pseudo-deleted. Actual cleaning up will not occur except during certain processes like before a page split.

You can calculate the percentage of RIDs that are pseudo-deleted based on the values of PSEUDO\_DEL\_ENTRIES and CARDF in SYSINDEXPART:

$(\text{PSEUDO\_DEL\_ENTRIES}/\text{CARDF}) * 100$

You can then determine if you should run REORG INDEX to physically remove the pseudo-deleted entries from the index.

To minimize the CPU cost of an index scan and reclaim space, it is important to remove pseudo-deleted entries. Every time a SQL statement makes a scan of an index, it has to scan all entries in the index, including pseudo-deleted entries that have not yet been removed.

We recommend that you run REORG INDEX if the percentage of pseudo-deleted entries is greater than 10%.

## 5.3 RUNSTATS

In this section, we discuss a number of topics related to collecting statistics with the RUNSTATS utility. RUNSTATS is the primary tool for providing information about distribution values for the DB2 optimizer.

### 5.3.1 Space statistics collected by RUNSTATS

Space statistics are captured when using RUNSTATS UPDATE ALL or UPDATE SPACE for the following DB2 catalog tables (Table 5-6 through Table 5-9 on page 115), with those common columns where space statistics are captured. The value for each of these statistics is -1 if statistics have not been gathered:

- SYSINDEXES and SYSINDEXES\_HIST

See Table 5-6.

Table 5-6 *SYSINDEXES and SYSINDEXES\_HIST*

Column name	Description
SPACEF	The number of kilobytes of DASD allocated to the index.

► SYSINDEXPART and SYSINDEXPART\_HIST

See Table 5-7.

Table 5-7 SYSINDEXPART and SYSINDEXPART\_HIST

Column name	Description
SPACEF	The number of kilobytes of DASD allocated to the index partition. The value is -1 if statistics have not been gathered.
DSNUM	The number of data sets.
EXTENTS	The number of data set extents.
FAROFFPOSF	Number of rows referenced far from the optimal position.
NEAROFFPOSF	Number of rows referenced near the optimal position.
LEAFDIST	100 times the number of pages between successive leaf pages.
LEAFNEAR	The number of leaf pages physically near a previous leaf page for successive active leaf pages.
LEAFFAR	The number of leaf pages located physically far away from previous leaf pages for successive (active leaf) pages accessed in an index scan. The value is -1 if statistics have not been gathered.
PSEUDO_DEL_ENTRIES	The number of pseudo-deleted entries in the index. These entries are marked as "logically" deleted, but still physically remain in the index. For a non-unique index, the value is the number of RIDs that are pseudo-deleted. For an unique index, the value is the number of both keys and RIDs that are pseudo-deleted.
PQTY	Primary space allocation in 4 KB blocks for the data set.
SQTY in SYSINDEXPART and SECQTYI in SYSINDEXPART_HIST	Secondary space allocation in 4 KB blocks for the data set.



► SYSTABLEPART and SYSTABLEPART\_HIST

See Table 5-8.

Table 5-8 SYSTABLEPART and SYSTABLEPART\_HIST

Column name	Description
CARDF	Number of rows.
DSNUM	Number of data sets.
EXTENTS	The number of data set extents.
FARINDREF	Number of rows relocated far from their original position.
NEARINDREF	Number of rows relocated near their original position.
PAGESAVE	Percentage of pages saved by data compression.
PERCACTIVE	Percentage of pages occupied by active pages.
PERCDROP	Percentage of pages occupied by pages from dropped objects.
PQTY	Primary space allocation in 4 KB blocks for the data set.
PSEUDO-DEL-ENTRIES	The number of pseudo-deleted entries in the index. These entries are marked as “logically” deleted, but still physically remain in the index. For a non-unique index, the value is the number of RIDs that are pseudo-deleted. For an unique index, the value is the number of both keys and RIDs that are pseudo-deleted.
SQTY in SYSTABLEPART and SECQTYI in SYSTABLEPART_HIST	Secondary space allocation in 4K B blocks for the data set.
SPACEF	The number of kilobyte DASDs allocated to the table space partition.

► SYSTABLES and SYSTABLES\_HIST

See Table 5-9.

Table 5-9 SYSTABLES and SYSTABLES\_HIST

Column name	Description
NPAGESF	The number of pages used by the table.
SPACEF	The number of kilobyte DASD allocated to the table.

## 5.3.2 When to run RUNSTATS

You need to run RUNSTATS when the underlying data significantly changes to data or indexes to warrant new statistics. This includes application updates and DDL changes. RUNSTATS collects information that can be used for access path selection by the DB2 optimizer or for space statistics. Space statistics consists of information surrounding space utilization and may provide enough details to determine when an object might benefit from a REORG, or when RUNSTATS or a COPY should be run. Space statistics is what we focus on in this section and not those statistics used by the DB2 optimizer. Refer to Chapter 11, “Gathering statistics” on page 291 for more information about the RUNSTATS utility.

Running inline statistics benefits REORG. You are already reading the data, so generating the statistics does not entail additional I/O. Similarly a LOAD REPLACE of a table space also affords you the option of using inline statistics.

Many sites opt to run RUNSTATS periodically, against objects where there is heavy inserts, deletes, and updates. There are also several RTS metrics that assist you in determining if your application would benefit from the RUNSTATS utility.

There are implications to running RUNSTATS and not rebinding static plans, so be prepared to consider the access path changes that may occur after a package rebind. Also, dynamic SQL may yield different access paths, as these new statistics can affect changes to the DB2 optimizer's choice.

RUNSTATS SHRLEVEL REFERENCE drains writers, but SHRLEVEL CHANGE runs like an application bound with ISOLATION(CS) and is the recommended utility when concurrency and high availability is necessary.

### **RUNSTATS TABLESPACE recommendation**

In regard to the question of when to run RUNSTATS TABLESPACE on an object, the recommendation is as follows:

- ▶ Run RUNSTATS whenever REORG or LOAD REPLACE or LOAD RESUME YES is performed on the object. You may use the inline statistic keyword STATISTICS and UPDATE option to update the statistics.
- ▶ Use the RTS estimate on the number of pages changed since last RUNSTATS. Use the information in STATSINSERTS, STATSDELETES, and STATSUPDATES to estimate the number of changed pages. Refer to 5.4, "What is new with real-time statistics (RTS)" on page 117 for additional information about RTS.
  - STATSINSERTS > 20%  
STATSINSERTS is the number of records inserted since the last RUNSTATS.
  - STATSDELETES > 20%  
STATSDELETES is the number of records deleted since the last RUNSTATS.
  - STATSUPDATES > 40%  
STATSUPDATES is the number of records updated since the last RUNSTATS.

### **RUNSTATS INDEX recommendation**

In regard to the question of when to run RUNSTATS INDEX on an object, the recommendation is as follows:

- ▶ STATSINSERTS > 20%  
STATSINSERTS is the number of records inserted since the last RUNSTATS.
- ▶ STATSDELETES > 20%  
STATSDELETES is the number of records deleted since the last RUNSTATS.

## **5.3.3 Frequently asked questions**

- ▶ You want to collect statistics but not affect any existing access paths. Can you collect statistics and have them stored in the catalog without affecting any binds/prepares?  
Yes. Specify UPDATE NONE HISTORY ALL or REPORT YES UPDATE NONE.

- Should you collect statistics on the DB2 Catalog?

Yes. It benefits any SQL that runs against the DB2 catalog. However, it does not benefit DB2 processing such as BIND or PREPARE because DB2 catalog access is completely determined by DB2.

- Is there any difference between running RUNSTATS TABLESPACE DB1.TS1 or RUNSTATS TABLESPACE DB1.TS1 TABLE(ALL)?

There is a big difference between the two! TABLE(ALL) collects column level statistics and column statistics for all columns and for all tables in the table space. Typically it uses more CPU and elapsed time. We recommend only collecting the column statistics needed for your application. You can use the features of Statistics Advisor (such as Optimization Service Center) to determine the best indexes for your application.

- Is there any difference between running RUNSTATS TABLESPACE DB1.TS1 INDEX (ALL) or RUNSTATS TABLESPACE DB1.TS1 RUNSTATS INDEX(ALL) TABLESPACE DB1.TS1?

No, they are semantically equivalent. In the first case, a single invocation of RUNSTATS does not allow for parallelism. In the second case, you trigger parallelism by running the two utilities in parallel jobs and this reduces overall elapsed time of the RUNSTATS utility.

- What is the difference between running RUNSTATS TABLESPACE DB1.TS1 TABLE(ALL) INDEX(ALL) or RUNSTATS TABLESPACE DB1.TS1 TABLE(ALL) RUNSTATS INDEX(ALL) TABLESPACE DB1.TS1?

In this case, they are functionally equivalent. However, with TABLE(ALL), column statistics are gathered for all columns in the table. With INDEX(ALL), column statistics are also gathered, but for all columns in the index. The big difference is that in the first case these index statistics are gathered from a table space scan, and in the second case the index statistics are gathered from an index scan. The index scan is less CPU intensive, so it is preferred.

## 5.4 What is new with real-time statistics (RTS)

Real-time statistics tables are a part of the DB2 catalog as of DB2 9 for z/OS. RTS keeps track of different metrics associated with partitions, table spaces, and index spaces with the intention that RUNSTATS need not be run as often. The DB2 optimizer only uses a few of the RTS values<sup>1</sup>. RTS is used for sort allocation as soon as UTSORTAL is set to YES, even if DB2 later decides to not allocate sort work data sets in DB2. You can benefit from improved estimates even when not exploiting the DB2 allocated sort work data sets. Refer to Chapter 6, “Sort processing” on page 141 for more information about the use of RTS.

There are actually close to 30 of these different space statistics; they are documented in *DB2 Version 9.1 SQL Reference*, SC18-9854. Among them, for example, there is a statistic for the last time an image copy was taken, and one for the number of rows inserted, updated, or deleted since the last utility (REORG or RUNSTATS or COPY) was executed.

There are two RTS tables named SYSIBM.SYSINDEXSPACESTATS and SYSIBM.SYSTABLESPACESTATS. There is one row in each table per partition. For non-partitioned table spaces and indexes, there is only one row. These tables reside in the table space SYSRTSTS and in the database DSNDB06.

<sup>1</sup> At the time of writing, the optimizer uses COLCARD, HIGHKEY, LOWKEY, HIGH2KEY, and LOW2KEY from SYSIBM.SYSCOLSTATS.

There are also three indexes defined on the tables DSNRTX01, DSNRTX02, and DSNRTX03. The index columns and its associated tables are documented in Table 5-10.

Table 5-10 RTS indexes

Table name	Index name	Index columns
SYSTABLESPACESTATS	DSNRTX01	DBID, PSID, PARTITION, and INSTANCE All in ascending order
SYSINDEXSPACESTATS	DSNRTX02	DBID, ISOBID, PARTITION, and INSTANCE All in ascending order
SYSINDEXSPACESTATS	DSNRTX03	CREATOR and NAME All in ascending order

RTS collects statistical information about each partition, table space, and index space. The type of information collected allows you to use it to determine when a reorganization of the data might be prudent, or when an image copy might be necessary, or when the statistical information collected by RUNSTATS might need updating.

Having the statistical information in RTS is extremely powerful. It allows you to monitor your objects based on the specific update activity metrics collected.

**Tip:** Use uncommitted read to minimize contention when accessing RTS tables.

### 5.4.1 When DB2 externalizes RTS

DB2 externalizes RTS in a number of different scenarios;

- ▶ When you issue STOP DATABASE(DSNDB06) SPACENAM(SYSRTSTS).

This command stops the RTS database and externalizes RTS for all objects. However, if activity is taking place, it would create an artificial gap in the collection of real-time statistics values, so from this point onwards, they are no longer accurate. If the table space is stopped only for a short period of time, the mismatch may not be significant, but it could accumulate over time if this is done regularly.

- ▶ When you issue START DATABASE(DSNDB06).

You would force an externalization of data by simply issuing START repeatedly without having to STOP the table space in between.

This command externalizes RTS only for database-name and space-name.

- ▶ At the end of the interval specified by the DSNZPARM STATSINT interval.
- ▶ Issue STOP DB2 MODE(QUIESCE).

DB2 writes RTS in memory to the statistics tables.

- ▶ During utility execution.

Refer to Table 5-12 on page 120 for additional information about each utility and what RTS columns are updated.

**Important:** DB2 does not maintain RTS for RTS objects. If you run a utility that includes RTS objects, RTS is not externalized for any of the objects in the utility list.

DB2 does not maintain incremental counters for RTS against SYSLGRNX and its indexes during utility execution. DB2 only maintains statistics for these objects during non-utility operations.

## 5.4.2 When are rows added/removed to/from RTS tables

A row is added or removed from the RTS tables when a certain DDL is issued. The different situations are summarized in Table 5-11.

*Table 5-11 How RTS tables are updated for DDL*

Operation	Result
CREATE TABLESPACE or INDEX	If TABLESPACE, create a row in SYSTABLESPACESTATS. If INDEX, create a row in SYSINDEXSPACESTATS. For partitioned tables, there is one row per partition. LOADLASTTIME set to CREATE timestamp. REORGLASTTIME set to CREATE timestamp. STATSLASTTIME set to NULL. COPYLASTTIME set to NULL. TOTALROWS set to zero. TOTALENTRIES set to zero. All other global counters set to NULL or a known value. Incremental counters set to zero.
DROP TABLESPACE or INDEX	If TABLESPACE, delete a row in SYSTABLESPACESTATS. If INDEX, delete a row in SYSINDEXSPACESTATS. For partitioned tables - there is one row per partition.

## 5.4.3 Establish a base value for RTS

For newly created objects, there is no need to establish a base value, as DB2 will maintain the counters right from the beginning. However, for objects that have existed before RTS were enabled, you need to run an appropriate utility to establish a base value for the RTS metrics (there are some basic metrics as well as special ones which can be used to determine when to run LOAD, REORG, and COPY). These base values are populated once you run a LOAD REPLACE, REORG, RUNSTATS, COPY, or a REBUILD INDEX. Table 5-12 on page 120 describes the actual resultant changes to RTS when these utilities complete successfully. Notice that each utility updates the associated timestamp and resets specific statistics depending on the particular utility.

For REORG, the RTS is updated before the LOG phase. Essentially the LOG phase does DML processing; during the LOG phase, INSERTs, UPDATEs, and DELETEs are recorded in RTS.

A new reference point is created with a REORG using inline statistics and an inline copy. Ideally, a new reference point should be established for all table spaces once the RTS objects are created. In DB2 9, this means after converting to NFM, it is best to establish a base point for all your objects.

Table 5-12 How different utilities affect RTS tables

Utility	SYSTABLESPACESTATS	SYSINDEXSPACESTATS
REORG	Set TOTALROWS, NACTIVE, SPACE, EXTENTS, REORGLASTTIME. Reset REORG related statistics: -REORGINSERTS -REORGDELETES -REORGUPDATES -REORGDISORGLGB -REORGUNCLUSTINS -REORGMASDELETE -REORGNEARINDREF -REORGFARINDREF Reset STATISTICS related statistics: -STATSLASTTIME, -STATSINSERTS, -STATSDELETES -STATSUPDATES -STATSMASDELETE Reset COPYDDN related statistics: -COPYLASTTIME -COPYUPDATEDPAGES -COPYCHANGES -COPYUPDATELRSN Note: Log apply changes for online REORG are treated as INSERT/DELETE/UPDATE.	Set TOTALENTRIES, NACTIVE, SPACE, EXTENTS, REORGLASTTIME. Reset REORG related statistics: -REORGINSERTS -REORGDELETES -REORGAPPENDINSERT -REORGPSEUDODELETES -REORGMASDELETE -REORGNEARINDREF -REORGFARINDREF -REORGNUMLEVELS Reset STATISTICS related statistics: -STATSLASTTIME, -STATSINSERTS, -STATSDELETES -STATSMASDELETE Reset COPYDDN related statistics: -COPYLASTTIME -COPYUPDATEDPAGES -COPYCHANGES -COPYUPDATELRSN -COPYUPDATETIME
RUNSTATS	Set STATSLASTTIME. Reset RUNSTATS related statistics: -STATSINSERTS -STATSDELETES -STATSUPDATES -STATSMASDELETE	May set: -STATSLASTTIME, -STATSINSERTS, -STATSDELETES, -STATSMASDELETE
COPY	Set COPYLASTTIME. Reset COPY related statistics: -COPYUPDATEDPAGES -COPYCHANGES -COPYUPDATELRSN -COPYUPDATETIME	Set COPYLASTTIME. Reset COPY related statistics: -COPYUPDATEDPAGES -COPYCHANGES -COPYUPDATELRSN -COPYUPDATETIME
LOAD REPLACE	Set TOTALROWS, NACTIVE, SPACE, EXTENTS, LOADLASTTIME, STATSLASTTIME (LOAD with STATISTICS option), COPYLASTTIME/COPYUPDATELRSN/COPYUPDATETIME (LOAD with COPYDDN). SET	Set TOTALENTRIES, NLEVELS, NACTIVE, SPACE, EXTENTS, LOADLASTTIME, STATSLASTTIME (LOAD with STATISTICS option), COPYLASTTIME/COPYUPDATELRSN/COPYUPDATETIME (LOAD with COPYDDN). SET
REORG PART	See REORG above.	Will not reset REORG statistics for NPIs. Statistics for NPIs are updated as INSERT/DELETE.

Utility	SYSTABLESPACESTATS	SYSINDEXSPACESTATS
LOAD REPLACE PART	See LOAD above.	Will not reset REORG statistics for NPIs. Statistics for NPIs are updated as INSERT and DELETE.
COPY with DSNUM option	Will not reset COPYLASTTIME. Will not reset COPY. Do not maintain statistics for a non-partitioned TS.	N/A.
RECOVER	To point_in_time: Reset REORG, STATS, COPY statistics to null. To CURRENT: Set NACTIVE, SPACE, EXTENTS.	To CURRENT: Set NACTIVE, SPACE, EXTENTS, NLEVELS.
REBUILD INDEX	N/A.	Set TOTALENTRIES, NLEVELS, NACTIVE, SPACE, EXTENTS, REBUILDLASTTIME. Reset REORG related statistics: -REORGINSERTS -REORGDELETES -REORGAPPENDINSERT -REORGPSEUDODELETES -REORGMASDELETE -REORGNARINDREF -REORGFARINDREF -REORGNUMLEVELS
LOAD RESUME SHRLEVEL CHANGE	Treated same as INSERT processing. See Table 5-13 on page 122.	N/A.

**Note:** No RTS data is collected for objects with the DEFINE NO option until the object is physically defined.

#### 5.4.4 Incremental updates to RTS

Once you have the base values established, the delta is recorded by DB2 and kept in memory. These memory based statistics are then written out to the RTS catalog tables depending on the DSNZPARM STATSINT (REAL TIME STATS). The default for STATSINT is 30 minutes. See Table 14-8 on page 402 for additional information.

Once the STATSINT timer occurs, these statistics are written to the RTS tables. RTS can also be updated when a pageset or partition is closed or at normal DB2 shutdown.

Different incremental statistics are added depending on the operation against the table. The different RTS statistics for specific SQL are documented in Table 5-13 for SYSIBM.SYSTABLESPACESTATS and in Table 5-14 for SYSIBM.SYSINDEXSPACESTATS.

Table 5-13 How SQL affects SYSTABLESPACESTATS

Operation	Result
INSERT	Increment REORGINSERTS, STATSINSERTS, TOTALROWS, and COPYCHANGES. May update NACTIVE, SPACE, EXTENTS, REORGUNCLUSTERINS, distinct NPAGES, COPYUPDATELRN, UPDATESTATSTIME, and DATASIZE.
UPDATE	Increment REORGUPDATES, STATSUPDATES, COPYUPDATEDPAGES, and COPYCHANGES counters. May update REORGNEARINDREF, REORGFARINDREF, NACTIVE, SPACE, EXTENTS for VARCHAR tables, distinct NPAGES, COPYUPDATELRN, UPDATESTATSTIME, and DATASIZE.
DELETE	Increment REORGDELETES, STATSDELETES, COPYCHANGES, and DATASIZE.
DELETE without the WHERE clause DROP TABLE for segmented TABLESPACEs	Increment the MASSDELETE/DROPs counter.
ROLLBACK - after INSERT	Decrement the INSERT counters.
ROLLBACK - after DELETE	Decrement the DELETE counters.
ROLLBACK - after UPDATE	Decrement the UPDATE counters.
ROLLBACK - after MASS DELETE	Will not decrement the MASSDELETES counter.
ROLLBACK - after DROP TABLE	Will not decrement the DROPs counter.
DB2 RESTART	Statistics counters will not be updated.
TRIGGER	May cause statistics to be updated for other objects.

Table 5-14 How SQL affects SYSINDEXSPACESTATS

Operation	Result?
INSERT	Increment REORGINSERTS, STATSINSERTS, and TOTALENTRIES. May update REORGAPPENDINSERT, REORGLFNEAR, REORGLFAR, REORGNLVL, NACTIVE, EXTENTS, SPACE, and NLEAF.
DELETE	Increment REORGDELETES and STATSDELETES. May update REORGMASSTDELETES, REORGPSEUDODELETES, and REORGNLVL.



Operation	Result?
INSERT for a COPY YES INDEX	In addition to INSERT, update COPYUPDATELRN, COPYUPDATEDPAGES, COPYCHANGES, and UPDATESTATSTIME.
DELETE for a COPY YES INDEX	In addition to INSERT, update COPYUPDATELRN, COPYUPDATEDPAGES, COPYCHANGES, and UPDATESTATSTIME.
DELETE without a WHERE clause	Increment REORGMASDELETE.
DROP TABLE	Increment REORGMASDELETE.
ROLLBACK - after INSERT	Decrement the INSERT counter.
ROLLBACK - after DELETE	Decrement the DELETE counter.
ROLLBACK - after UPDATE	Decrement the UPDATE counter.
ROLLBACK - after MASS DELETE	Will not decrement the MASSDELETES counter.
ROLLBACK - after DROP TABLE	Will not decrement the DROPS counter.
DB2 RESTART	Statistics counters will not be updated.

**Note:** In data sharing, each member collects its own statistics and writes to the RTS tables. In addition, each member can specify its own statistics interval.

For information about important topics related to RTS and data sharing, refer to *DB2 Version 9.1 for z/OS Utility Guide and Reference*, SG18-9855.

### 5.4.5 RTS manager

The component of DB2 responsible for RTS functions is the RTS manager. The RTS manager runs as a subtask in the DBM1 address space and is created during the initialization of DB2. The RTS manager is responsible for the externalization of the in-memory statistics to the RTS tables.

The DBM1 address space houses RTS blocks used for each object where statistical information is kept. The size of the RTS blocks is approximately 140 bytes per pageset or partition and is kept above the bar. Each block is allocated when a pageset/partition is opened for update at the table space level. Because the metric SYSINDEXSPACESTATS.LASTUSED is collected for an index, space is allocated for indexes when an index is opened. These blocks are freed when a pageset or partition is closed and the statistics collected are written to the RTS tables.

The RTS manager ensures its active statistics blocks are kept in clustering order and all inserts and updates to the rows in the RTS tables are accomplished via the clustering index. Refer to Table 5-10 on page 118 for a description of the indexes on the RTS tables.

## 5.4.6 New RTS columns

There are a few new columns added to the DB2 9 RTS tables that are worth mentioning here. Table 5-15 contains the new columns from SYSIBM.SYSTABLESPACESTATS and Table 5-16 contains the ones from SYSIBM.SYSINDEXSPACESTATS.

Table 5-15 New SYSTABLESPACESTATS columns

Column name	Description
DATASIZE	The total number of bytes that row data occupies in the data rows or LOB rows.
INSTANCE	Indicates if the object is associated with data set instance 1 or 2.
NPAGES	The number of distinct pages with active rows in the partition or table space.

Table 5-16 New SYSINDEXSPACESTATS columns

Column name	Description
INSTANCE	Indicates if the object is associated with data set instance 1 or 2.
LASTUSED	The date when the index is used for SELECT, FETCH, searched UPDATE, searched DELETE, or used to enforce referential integrity constraints. The default value is 1/1/0001.
NAME	The name of the index.
NLEAF	The number of leaf pages.
NPAGES	The number of distinct pages with active rows in the associated table.

## 5.4.7 Accuracy of RTS

The RTS values are always delayed by the timer interval specified in DSNZPARM as STATSINT. In addition, there are several cases where DB2 is unable to externalize RTS in memory to the RTS tables. You should be aware of these different scenarios:

- ▶ DB2 abend  
All in-memory RTS values may be completely lost if DB2 abends.
- ▶ STOP DB2 MODE(FORCE)  
All in-memory RTS values may be completely lost if a STOP DB2 MODE(FORCE) is issued.
- ▶ STOP DATABASE(DSNDB06) SPACE(SYSRTSTS)  
DB2 may be unable to externalize RTS data if a STOP DATABASE(DSNDB06) SPACE(SYSRTSTS) is issued.
- ▶ Replace a pageset using a copy taken by DSN1COPY.  
Refer to “DSN1COPY considerations” on page 125.
- ▶ Utility restart  
Certain utility restart scenarios might affect the accuracy of in-memory RTS.

► Third-party utilities

Run third-party utilities after flushing in-memory RTS. Update RTS after running third-party utilities. Otherwise, you run the risk of inaccurate RTS.

**Important:** In-memory RTS can be forced to the RTS tables by issuing a `START DATABASE(dbname) TSNAME(tsname)` command for the objects in `dbname.tsname`. This will work even if the object is already in R/W status.

### DSN1COPY considerations

If you replace DB2 objects outside of DB2's control, then you may need to be concerned about the accuracy of the RTS for these objects. For example, suppose you use a DSN1COPY version of an object to replace it; the RTS may not accurately reflect the current statistics. In order to inform DB2 of its correct statistics, you may have to manually update them. Otherwise, you run the risk of possibly generating an incorrect sort work space. For example, if you replace a table that had 10,000 rows with a DSN1COPY version that has 1,000,000 rows, your REORG might fail because the RTS TOTALROWS may be set to 10,000.

As an example, consider setting TOTALROWS in SYSIBM.SYSTABLESPACESTATS or TOTALENTRIES in SYSIBM.SYSINDEXSPACESTATS to NULL. This allows you to invalidate the existing statistics or, alternatively, you can set these statistics to the actual number of rows and or keys if it is known. Of course, if you are replacing the object with significantly different data and you do not know the precise statistics associated with the DSN1COPY version, then setting these statistics to null is the best choice you may have.

## 5.4.8 Considerations on RTS usage

In this section, we discuss the new statistical index tracking function, LASTUSED, and provide scenarios of RTS utilization. RTS tables are just like other application objects. They need to be copied, reorganized, and have RUNSTATS generate space and access path statistics against them.

### Index usage tracking

DBAs often want to track when an index was last used by an application, especially to find those indexes that are not being used at all. The SYSINDEXSPACESTATS column LASTUSED, introduced with DB2 9, represents the date when the index was last referenced. This metric is maintained whenever a SELECT, FETCH, searched UPDATE, or a searched DELETE SQL statement is issued against the object. Also, if an object uses RI and DB2 has to enforce referential integrity constraints with either dynamic or static SQL that includes this object, its LASTUSED column is also updated.

In effect, LASTUSED allows you to determine what indexes have not been used over a period of time. DBAs may then use this information to decide to eliminate the index. Extra indexes lead to longer UPDATE, DELETE, and INSERT activity against the base table. So, deleting these underutilized indexes can save valuable CPU and elapsed time for an application, especially if the index has never been considered by the DB2 optimizer or used by any dynamic SQL running against the object.

**Note:** The existence of an index has the opportunity for RUNSTATS to collect statistics (by default) that would not have been collected otherwise. Dropping an index that appears to not have been used for a long time may affect the access path of dynamic SQL. Although there is no apparent dependency on the index itself, the statistics related to the columns in the index may be used by the optimizer when deciding on an access path.

Evaluate the implications of dropping an index with a decision based only on LASTUSED input.

The LASTUSED metric defaults to a value of 01-01-0001. If an index exists with this default date, then it has never been used since moving to DB2 9 NFM.

**Note:** If an index changes because of an INSERT or UPDATE, LASTUSED is not updated.

### Guidelines when accessing RTS

If you are processing RTS tables directly, be aware of possible contention with the RTS manager.

There are a few guidelines for SQL and utilities when accessing RTS objects to avoid contention:

- ▶ Avoid timeouts or deadlocks with the RTS manager.
  - Use UR lock isolation level when accessing RTS tables.
  - Use SHRLEVEL CHANGE when running REORG, RUNSTATS, or COPY against the RTS objects.
- ▶ Do not mix RTS objects with other objects in a utility list.

If RTS is mixed in a list with other objects, RTS will not be reset for other objects in the list.
- ▶ Disaster recovery
  - Recover RTS objects after DB2 catalog and directory objects are recovered.
  - Explicitly issue START DATABASE(DSNRTSDB) after RTS objects are recovered.

### Maintaining an historical view of RTS

There is no historical capability with the current implementation of RTS. If you want to maintain an historical view of your data, you have to develop a manual process.

You can easily create the same RTS tables in your own database and add a **TIMESTAMP** column to keep track of these RTS values over time. A possible method is provided in Example 5-2, which gives a sample DDL that can be used to create a copy of the RTS tables.

---

*Example 5-2 Creating historical RTS tables*

---

```
CREATE TABLE TABLESPACESTATS_HIST LIKE SYSIBM.SYSTABLESPACESTATS;
COMMIT;
ALTER TABLE TABLESPACESTATS_HIST
    ADD COLUMN CAPTURE_TIME TIMESTAMP NOT NULL WITH DEFAULT
    ;
COMMIT;
CREATE TABLE INDEXSPACESTATS_HIST LIKE SYSIBM.SYSTABLESPACESTATS
    ;
COMMIT;
ALTER TABLE INDEXSPACESTATS_HIST
    ADD COLUMN CAPTURE_TIME TIMESTAMP NOT NULL WITH DEFAULT
    ;
COMMIT;
```

---

You can then take the actual RTS values and insert them into the historical tables. One approach could be using the cross loader capability. Here in Example 5-3 we load all the rows from the **SYSIBM.SYSTABLESPACESTATS** into the **TABLESPACESTATS\_HIST** table. Eventually, you would want to choose only recently updated rows that say “WHERE **UPDATESTATSTIME - CURRENT\_TIME < 24hours**”. You need to decide how much historical data you keep and how often you collect it. You can collect it for every **STATSINT**, once a week, or maybe decide to collect it for some time in between these two extremes.

---

*Example 5-3 Cross loader for RTS history*

---

```
/*
/* CROSS LOAD WITH REPLACE
/*
//LOAD EXEC PGM=DSNUTILB,PARM='DB9A,MARY00'
//STEPLIB DD DSN=DB9A9.SDSNEXIT,DISP=SHR
//          DD DSN=DB9A9.SDSNLOAD,DISP=SHR
//SYSPRINT DD SYSOUT=*
//UTPRINT DD SYSOUT=*
//SYSIN DD *
EXEC SQL
    DECLARE C1 CURSOR FOR SELECT * FROM "SYSIBM"."SYSTABLESPACESTATS"
    ORDER BY DBID,
            PSID,
            PARTITION,
            INSTANCE
ENDEXEC
LOAD DATA INCURSOR C1 LOG NO RESUME YES
    INTO TABLE "DB2R8"."TABLESPACESTATS_HIST"
    SHRLEVEL NONE
/*
```

---

Periodically, you may need to insert those rows that are not already in these historical tables. One possibility to find those objects is to code a subselect against the RTS tables to find these rows that exist in RTS but not in the history table. See Example 5-4 for sample SQL that allows you to find these tables.

*Example 5-4 Objects in SYSTABLESPACESTATS but not in TABLESPACESTATS \_HIST*

---

```
SELECT P.DBNAME, P.TSNAME, P.PARTITION
FROM   SYSIBM.SYSTABLESPACE S
INNER JOIN SYSIBM.SYSTABLEPART P
ON P.DBNAME = S.DBNAME
ANDP.TSNAME = S.NAME
WHERE NOT EXISTS
      (SELECT 1
       FROM TABLESPACESTATS _HIST T
       WHERE S.DBID=T.DBID
       AND   S.PSID=T.PSID
       AND   P.PARTITION=T.PARTITION)
ORDER BY P.DBNAME, P.TSNAME, P.PARTITION
FOR FETCH ONLY WITH UR;
```

---

### **Finding objects that do not exist in RTS tables**

Use the SQL provided in Example 5-5 to find those objects that do not exist in SYSTABLESPACESTATS and then run a REORG TABLESPACE on each of them to establish a base point for that object.

*Example 5-5 Objects that do not exist in SYSTABLESPACESTATS*

---

```
SELECT P.DBNAME, P.TSNAME, P.PARTITION
FROM   SYSIBM.SYSTABLESPACE S
INNER JOIN SYSIBM.SYSTABLEPART P
ON P.DBNAME = S.DBNAME
ANDP.TSNAME = S.NAME
WHERE NOT EXISTS
      (SELECT 1
       FROM SYSIBM.SYSTABLESPACESTATS T
       WHERE S.DBID=T.DBID
       AND   S.PSID=T.PSID
       AND   P.PARTITION=T.PARTITION)
ORDER BY P.DBNAME, P.TSNAME, P.PARTITION
FOR FETCH ONLY WITH UR;
```

---

Similarly, use the SQL provided in Example 5-6 to find those objects that do not exist in SYSINDEXSPACESTATS. You could then run a REBUILD INDEX on each of them.

*Example 5-6 Objects that do not exist in SYSINDEXSPACESTATS*

---

```
SELECT P.IXCREATOR,
       P.IXNAME,
       P.PARTITION
FROM   SYSIBM.SYSINDEXES X
INNER JOIN SYSIBM.SYSINDEXPART P
ON     P.IXCREATOR = X.CREATOR
AND    P.IXNAME = X.NAME
WHERE  NOT EXISTS
      (SELECT 1
       FROM SYSIBM.SYSINDEXSPACESTATS T
       WHERE X.DBID=T.DBID
       AND   X.ISOBID=T.ISOBID
       AND   P.PARTITION=T.PARTITION)
ORDER BY P.IXCREATOR, P.IXNAME, P.PARTITION
```

---

### Relationship of RTS with sort

For some sorts, DB2 uses RTS to estimate the number of records to be sorted. Before using RTS, the information that was used was based on space statistics in the DB2 catalog. Only if RTS is not available does DB2 revert back to using the DB2 catalog statistics.

As mentioned earlier, the required RTS values are initialized by REORG TABLESPACE, LOAD REPLACE, and REBUILD INDEX. If RTS is not available for an object, some fallback logic exists in the following cases:

- ▶ Use index/partition RTS instead of table space/partition RTS.
- ▶ Use table space/partition RTS instead of index/partition RTS.
- ▶ Use the sum of one index for each table in table space for table space RTS.

**Important:** Fallback logic is not used for XML objects, as there is not a 1:1 relationship between index and table space.

Refer to Chapter 6, “Sort processing” on page 141 for more information about sort topics.

## 5.4.9 Recommendations

In this section, we describe some general recommendations concerning when to run a REORG utility. Unfortunately, there is no one absolutely reliable statistic that applies in all cases as to when to reorganize a table space or an index. However, understanding the different types of data disorganization metrics will aid you in determining a reorganization strategy for your company’s needs. You can refine it over time based on your specific situation and application needs.

In this section, we discuss specifics related to the REORG TABLESPACE and REORG INDEX utility.

### When to run REORG TABLESPACE

We show criteria based on RTS, RUNSTATS space statistics, and ALTER related collected information for tables spaces.

### ***When to run a REORG based on RTS for table space***

In this section, we examine the usefulness of the RTS columns contained in SYSIBM.SYSTABLESPACESTATS.

For non-LOB table spaces, use the following recommendations:

You should consider running REORG TABLESPACE in the following situations and only when NPAGES>5. In addition, pay attention to the criteria that the table space is worth spending the time on a REORG, so include only those objects where TOTALROWS > 100 or CARDF > 100 or NACTIVE > 100 before examining any other RTS criteria.

- ▶ REORGUNCLUSTINS (number of records inserted since the last REORG that are not well-clustered)/TOTALROWS > 10%

REORGUNCLUSTINS is irrelevant if your access is predominantly random using an index. REORGUNCLUSTINS is an indication of insert behavior and is correlated to the cluster ratio only if there are no updates or deletes.

REORGUNCLUSTINS is incremented when inserting a record no more than 16 pages away from the index manager candidate page. This counter is not incremented for member cluster objects (independent of freespace) or for append operations. The value is only incremented when the index manager provides a candidate page.

For an example of using this criteria, look at the SQL provided in Example 5-7. An example using DSNACCOX with a similar formula is illustrated in Figure 5-8 on page 110.

#### ***Example 5-7 SQL using REORGUNCLUSTERINS/TOTALROWS***

---

```
-- RTS TO DETERMINE IF REORG IS REQUIRED
--
SELECT DBNAME
       , NAME
       , PARTITION
       , DEC(REORGUNCLUSTINS / (TOTALROWS+REORGDELETES) * 100.0,15,3)
         AS PCTUNCL
       , DEC(REORGUNCLUSTINS) AS UNCLUSTINS
       , DEC(TOTALROWS) AS TOTALROWS
       , REORGLASTTIME
       , RS.*
FROM SYSIBM.TABLESPACESTATS RS
WHERE REORGUNCLUSTINS / TOTALROWS > .10
      AND TOTALROWS > 0
      AND REORGUNCLUSTINS > 0
ORDER BY 4 DESC
WITH UR;
```

---

- ▶ (REORGNEARINDREF+REORGFARINDREF)/TOTALROWS > 5% in data sharing, >10% in non-data sharing

REORGNEARINDREF+REORGFARINDREF is the number of overflow rows since the last REORG



► REORG UPDATE Statistics

Having > 25% INSERTs, UPDATEs, or DELETEs is intended to provide a more preventive check. For example, even if reviewing the access path and the average CPU and elapsed time for an SQL statement does not indicate a problem, there maybe a problem that will show up in the future. It is even possible that a problem already exists, but it does not show up yet, based on the limited set of information you may have gathered. Some examples specific to REORG include running out of free space so that massive index splits or disorganization is imminent, and reducing excessive reclaimable but not reclaimed space such that an overall data set size is kept under control.

Here we provide the generalizations predicting when to REORG based on the update activity of the underlying object:

- REORGININSERTS/TOTALROWS > 25%

REORGININSERTS is the number of records inserted since the last REORG.

- REORGDELETES/TOTALROWS > 25%

REORGDELETES is the number of records deleted since the last REORG.

- REORGMASDELETE > 0

REORGMASDELETE is the number of mass deletes from a segmented or LOB table space or the number of dropped tables from a multi-table table space since the last REORG or LOAD REPLACE.

► EXTENTS (number of extents) > 50

EXTENTS is the current number of extents. The number of extents used to be a performance concern. These days, with the different and much improved storage managers, performance is not the major concern. Availability is what drives this metric. You do not want to get close to the maximum number of extents and run the risk of not having a table or an index available. Monitor the number of extents accordingly.

The most important consideration here is the size of extent, not necessarily the number of extents. If there are many extents, for example, greater than 10, with a small extent size, for example, one track, there is a significant performance penalty for each extent as well as for a query reading through multiple extents. In this case, having fewer but bigger extents is preferable.

On the other hand, if there are many extents with large extent size, for example, 100 cylinders each, the performance impact of having many extents is negligible. So for a multi-extent data set in general, a secondary extent size of greater than 10 MB, or 15 cylinders, is recommended. As the data set size becomes bigger, a secondary extent size should also be increased in size.

The EXTENTS value is useful in tracking when the allocation quantity may need to be changed to prevent extend failures as the maximum number of extents is approached prematurely relative to the maximum data set size. In DB2 V8, the managed extent size automatically adjusts an extent size to avoid reaching a VSAM maximum extent limit of 255 prematurely before reaching the maximum data set size.

► SPACE > 2 \* (DATASIZE / 1024)

When free space is more than used space, you may want to run a REORG.

► REORGMASDELETE > 0

REORGMASDELETE is the number of mass deletes on a segmented table space and the number of dropped tables in a multi-table table space since the last REORG.

**Important:** In the case of a REORG against the DB2 Catalog, double the thresholds given above, because of the less sensitivity and additional cost for the links contained in the DB2 Catalog.

### ***When to run a REORG based on RUNSTATS***

This section uses those statistics collected by the RUNSTATS utility to determine when to REORG. In general, start using RTS as a more reliable and accurate method of determining when to run a REORG TABLESPACE.

For non-LOB table spaces, use the following recommendations:

- ▶ PERCDROP > 10%

PERCDROP is the percentage of unused space in a non-segmented table space. This space occupied by rows from dropped tables in non-segmented non-partitioned table spaces is not reclaimed until you reorganize the table space. Space in a segmented table space is reusable right after a commit.

- ▶  $(\text{NEARINDREF} + \text{FARINDREF}) / \text{CARDF} > 10\%$  non-data-sharing, > 5% if data sharing

NEARINDREF and FARINDREF are discussed in more detail in 5.2.3, “NEARINDREF and FARINDREF” on page 99.

- ▶ FAROFFPOSF / CARDF > 10%

FAROFFPOSF is discussed in more detail in 5.2.2, “NEAROFFPOSF, FAROFFPOSF, and CARDF” on page 97.

- ▶ CLUSTERRATIOF < 90%

If the index is a clustering index, then CLUSTERRATIOF is the percentage of pages in clustering order. Remember this metric is irrelevant if you predominately access the data randomly via an index.

CLUSTERRATIOF is discussed in more detail in 5.2.1, “CLUSTERRATIOF and DATAPEATFACTOR” on page 95.

### ***When to run a REORG for LOB table spaces***

This section includes combined recommendations for running a REORG against a LOB table space using both RTS and RUNSTATS.

For LOB table spaces, use the following recommendations:

- ▶ SYSIBM.SYSLOBSTATS.ORGRATIO < 10% (<V9) or 80%(=V9)

SYSIBM.SYSLOBSTATS.ORGRATIO is the percentage of organization in the LOB table space. A value of 100 indicates perfect organization and a value of 1 means totally disorganized.

The recommendation changes depend on what version of DB2 you are using. Use 10% if you are using DB2 V8 and 80% for DB2 V9.

If this metric is satisfied, then consider the following items and use this criteria in addition to the one above:

- REORGDISORGLob/TOTALROWS > 50%  
REORGDISORGLob is the number of LOBs inserted since the last REORG that are not perfectly chunked.
- SYSTABLEPART.SPACEF (in KB) >  
 $2 * \text{SYSTABLEPART.CARDF} * \text{SYSLOBSTATS.AVGSIZE} / 1024$  (DB2 V9)
- SYSTABLESPACESTATS.SPACE >  $2 * \text{DATASIZE} / 1024$  (DB2 V9)
- SYSLOBSTATS.FREESPACE (in KB)/SYSTABLESPACE.NACTIVE\*PGSIZE (in KB) > 50%

### **Other**

If a table space is in a restrictive status, then a REORG TABLESPACE is advised:

- ▶ REORP (REORG PENDING)
- ▶ AREO\* (Advisory REORG pending) as result of an ALTER TABLE statement
- ▶ ARBDP (advisory REBUILD pending) as a result an ALTER statement

### **When to run REORG INDEX**

Here we show criteria based on RTS, RUNSTATS, and ALTER related collected information.

#### ***When to run a REORG INDEX based on RTS***

These recommendations are also useful for LOB auxiliary indexes. Consider running REORG INDEX when SYSINDEXSPACESTATS.NLEAF > 5 in the following cases when using RTS from the table SYSINDEXSPACESTATS. Also consider using the following recommendations when the RTS metrics or TOTALROWS, CARDF, or NACTIVE > 100:

- ▶ REORGPSEUDODELETES/TOTALENTRIES > 10% in non-data sharing, 5% if data sharing  
REORGPSEUDODELETES is the number of index entries pseudo-deleted since the last REORG. In data sharing, a pseudo-deleted entry can cause a S-lock/unlock in INSERT for a unique index.
- ▶ REORGLEAFFAR/NACTIVE > 10%  
REORGLEAFFAR is the net number of leaf pages located physically far away from previous leaf pages for successive active leaf pages that occurred since the last REORG, REBUILD INDEX, or LOAD REPLACE.  
NACTIVE is the number of active pages in the index space or partition.
- ▶ REORGININSERTS/TOTALENTRIES > 25%  
REORGININSERTS is the number of index entries inserted since the last REORG.
- ▶ REORGDELETES/TOTALENTRIES > 25%  
REORGDELETES is the number of index entries deleted since the last REORG.
- ▶ REORGAPPENDINSERT / TOTALENTRIES > 20%  
REORGAPPENDINSERT is the number of index entries that have a key value that is greater than the maximum key value in the index or partition that have been inserted since the last REORG, REBUILD INDEX, or LOAD REPLACE on the index space of partition.
- ▶ EXTENTS > 50  
EXTENTS is the number of extents. This is the same recommendation as for REORG TABLESPACE in “When to run a REORG based on RTS for table space” on page 130.

### ***When to run a REORG INDEX based on RUNSTATS***

These statistics are available but not recommended for use with DB2 9. It is much better to use RTS. These recommendations are also useful for LOB auxiliary indexes.

- ▶ LEAFFAR / NLEAF > 10%

The metric LEAFFAR is discussed in 5.2.7, “LEAFNEAR and LEAFFAR” on page 107.

- ▶ PSEUDO\_DEL\_ENTRIES / CARDF > 10% in non-data sharing and > 5% in data sharing

The metric PSEUDO\_DEL\_ENTRIES is discussed in Table 5-7 on page 114.

CARDF is the total number of rows.

### ***Other cases***

If an index is in AREO\* status (advisory REORG pending) or ARBDP (advisory REBUILD pending) as the result of an ALTER statement, then we recommend running a REORG INDEX.

### **DB2 Administration Tool to access RTS**

Refer to “DB2 Administration Tool” for information about using the DB2 Administration tool to access RTS data and to generate different batch jobs depending on the results (COPY, REORG, RUNSTATS, or a resize operation). It can help determine when to run these utilities based on different criteria you specify.

## **5.5 Trends from historical statistics**

There are a number of historical statistical tables in catalog DSNDB06 (table space SYSHIST). These tables contain all statistical fields of the “parent” catalog tables with date and timestamp. The historical statistical data can be used to set thresholds and trigger utilities. In this section, we discuss the following:

- ▶ Monitoring space growth of table space and index space
- ▶ Compression dictionary considerations

### **5.5.1 Monitoring space growth**

Table SYSTABLEPART\_HIST has two fields, CARDF and SPACEF, which contain the values for total number of rows and total amount of disk space allocated to the table space or partition space, respectively. Similar fields also exist for index space in SYSINDEXPART\_HIST.

CARDF and SPACEF can be tracked regularly for space growth of an object. Assuming constant growth over time, these numbers can be used to derive growth trends to plan for future needs. Consider the following sample SQL:

```
SELECT MAX(CARDF), MIN(CARDF), ((MAX(CARDF)-MIN(CARDF))*100)/MIN(CARDF),  
      (DAYS(MAX(STATSTIME))-DAYS(MIN(STATSTIME)))  
FROM SYSIBM.SYSTABLEPART_HIST  
WHERE DBNAME='DB' AND TSNAME='TS';
```

Assuming that the number of rows is constantly increasing, so that the highest number is the latest, the query shows the percentage of rows added over a specific time period. This could be extrapolated to scale on a monthly or yearly basis.

## COPY utility

The space growth and SYSCOPY entries of a DB2 object can be used in determining the requirement for an image copy. If the space growth is greater than a predetermined percentage growth value since the last full image copy, then the COPY utility can be triggered to take a full image copy of the object.

### 5.5.2 Compression dictionary considerations

When COMPRESS YES is set for a table space, its compression dictionary is initially built either via the LOAD or REORG utility. The PAGESAVE field in SYSTABLEPART and SYSTABLEPART\_HIST keeps track of the percentage of pages saved due to the compression. The performance of the compression and decompression dictionary depends on the change of data values for update activity on the table space. Over a time period, the PAGESAVE can be reduced below a *threshold\_limit* value, and the DBA needs to rebuild the dictionary.

Regularly building the dictionary is expensive, and it should be avoided. The PAGESAVE value in SYSTABLEPART\_HIST can be used to monitor the effectiveness of the dictionary. If the following situation occurs:

$$(\max(\text{pagesave}) - \min(\text{pagesave})) * 100 / \max(\text{pagesave}) > \text{threshold\_limit}$$

you should remove the keyword KEEPDICTIONARY from the REORG or LOAD utility. The utility will then build a new dictionary. We recommend a *threshold\_limit* of 10%.

## 5.6 Stored procedure DSNACCOX

DSNACCOX is a DB2-supplied stored procedure that allows you to query the RTS tables. It is an enhancement over the previous version DSNACCOR, which includes new records, the ability to access the new columns in RTS, and new formulas to help you determine for a particular object when to run a REORG, RUNSTATS, or COPY based on different pre-supplied formulas. These recommendations are provided back to the calling program in a result set from the stored procedure.

You have the option to choose a predefined formula for these recommendations based on your site's preferences. In addition, you can find those objects that exceed a predefined number of extents or those objects in a restricted state, such as COPY pending.

There is also an exception table that can be used to explicitly specify those objects that you may not want to REORG or must REORG on a regular basis. To prevent DSNACCOX from triggering because of them, identify such objects and put them in an exception list.

**Tip:** Bind the package associated with DSNACCOX with ISOLATION(UR) to minimize lock contention against the RTS objects.

Formulas are provided in the following useful situations where maintenance for the underlying objects might be recommended:

- ▶ Full image copy on a table space
- ▶ Full image copy on an index space
- ▶ Incremental image copy on a table space
- ▶ REORG on a table space
- ▶ REORG on an index space
- ▶ RUNSTATS on a table space
- ▶ RUNSTATS on an index space
- ▶ Excessive extents
- ▶ Objects in a restricted state

For more details about the stored procedure DSNACCOX or on the various formulas used for these situations, see *DB2 Version 9.1 Performance Monitoring and Tuning Guide*, SC18-9851. These formulas may not be applicable to all situations, but they do give you a good starting point.

The DB2 Administration Tool uses these formulas and you can change the different values in the Performance Queries options 14 and 14X. For more information about this topic, refer to “DB2 Performance Queries” on page 423.

A similar formula to the one used in Example 5-7 on page 130 is given here; this formula is embedded in DSNACCOX.

*Example 5-8 DSNACCOX formula for recommending a REORG on a table space*

---

```
((QueryType='REORG' OR QueryType='ALL') AND
(ObjectType='TS' OR ObjectType='ALL')) AND
(REORGLASTTIME IS NULL AND LOADRLASTTIME IS NULL) OR
(NACTIVE IS NULL OR NACTIVE > 5) AND
((((REORGINSERTS×100)/TOTALROWS>RRTInsertPct) AND
REORGINSERTS>RRTInsertAbs) OR
(((REORGDELETE×100)/TOTALROWS>RRTDeletePct) AND
REORGDELETE>RRTDeleteAbs) OR
(REORGUNCLUSTINS×100)/TOTALROWS>RRTUnclustInsPct OR
(REORGDISORGLob×100)/TOTALROWS>RRTDisorgLOBPct OR
(SPACE×1024)/DATASIZE>RRTDataSpaceRat OR
((REORGNEARINDREF+REORGFARINDREF)×100)/TOTALROWS>RRTIndRefLimit OR
REORGMASDELETE>RRTMassDelLimit OR
EXTENTS>ExtentLimit)) OR
((QueryType='RESTRICT' ORQueryType='ALL') AND
(ObjectType='TS' OR ObjectType='ALL') AND
The table space is in advisory or informational reorg pending status))
```

---

## 5.7 DB2 Administrative Task Scheduler

The DB2 Administrative Task Scheduler gives you the capability to schedule different tasks under the auspices of DB2. It starts as a task automatically when DB2 9 is started.

The DB2 Administrative Task Scheduler has its own address space, so if you have DB9AMSTR, DB9ADBM1, DB9ADIST, and DB9AIRLM, you will also see DB9AADMT.

The architecture of the DB2 Administrative Task Scheduler is provided in Figure 5-9. Within DB2, you specify the name of the started task using the parameter ADMTPROC. This parameter represents the administrator scheduler address space startup procedure. When the DB2AMSTR address space starts, it also starts the administrative scheduler address space, DB2AADMT. A user accesses the administrative scheduler via SQL using DB2 supplied stored procedures and user-defined table functions.

You can add tasks and remove them via these stored procedures. You can use the user defined table functions to list the scheduled tasks and review their execution status. The administrative tasks are stored in a DB2 table and also in a VSAM data set for consistency purposes. So if DB2 is down, the scheduler can still monitor its jobs and run based on information stored in the VSAM data set. Later on, when DB2 is functional again, the two repositories are synchronized.

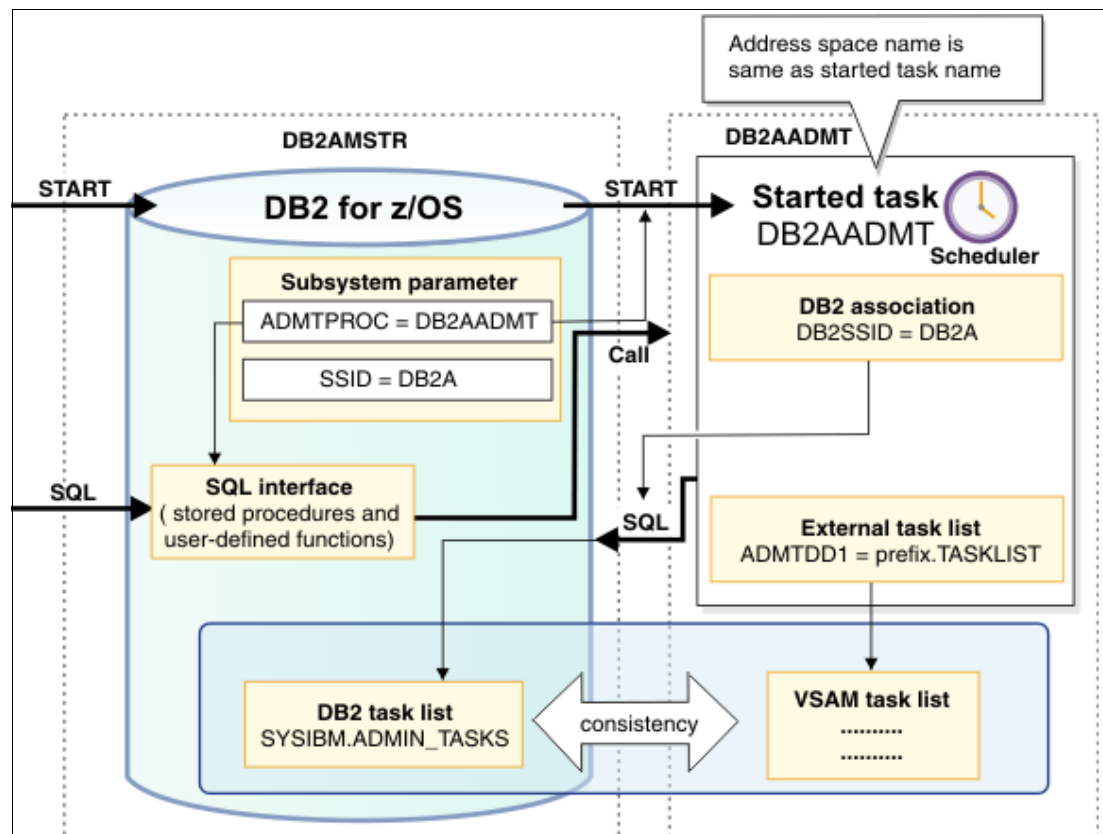


Figure 5-9 Administrative Task Scheduler

Table 5-17 provides a list of the DB2 objects associated with the Administrative Task Scheduler.

*Table 5-17 DB2 objects used by the Administrative Task Scheduler*

Object type	Object name	Object description
Database	DSNADMDB	Main database.
Table space	DSNADMTS	Main table space.
Table	ADMIN_TASKS	Main table
Index	ADMIN_TASKS_IX	Index on TASK_ID.
Stored procedure	ADMIN_TASK_ADD	Adds a task to ADMIN_TASKS.
	ADMIN_TASK_REMOVE	Deletes a task from ADMIN_TASKS.
User-defined table functions	ADMIN_TASK_LIST	Generates a list of tasks.
	ADMIN_TASK_STATUS	Views the status of all tasks.

For more information about the DB2 Administrative Task Scheduler and details about how to submit tasks like utility jobs via the DB2 Administrative Task Scheduler, refer to *DB2 Version 9.1 Administration Guide*, SC18-9840.

## 5.8 IBM DB2 Tools

There are a number of DB2 Tools that can assist you with utility generation, utility function, and some special functions related to DB2 catalog statistics. In this section, we discuss some of these features from a high level.

In this book, we focus on the capabilities of each of the mentioned tools keeping DB2 utilities as our primary focus. The tools listed here are briefly described and are as follows:

- ▶ DB2 Administration Tool
- ▶ DB2 Utility Enhancement Tool
- ▶ DB2 Automation Tool
- ▶ DB2 High Performance Unload for z/OS
- ▶ DB2 Recovery Expert for z/OS
- ▶ DB2 Change Accumulation Tool
- ▶ DB2 Storage Management Utility
- ▶ Optim Query Tuner for DB2 for z/OS and Optim Query Workload Tuner for DB2 for z/OS
- ▶ DB2 Buffer Pool Analyzer

More information about each of these tools can be found in Appendix A, “DB2 Tools” on page 413 and at the following address:

<http://www.ibm.com/software/data/db2imstools/products/db2-zos-tools.html>



## 5.8.1 DB2 Administration Tool

The DB2 Administration Tool allows you to perform specific functions in conjunction with all of the DB2 utilities. This tool serves many purposes: catalog navigation, migration of objects, generates DDL, change management, and so on. From a DB2 utility perspective, the following capabilities exist to help DBAs with their productivity in their day to day interaction with DB2:

- ▶ Generate LISTDEFs and TEMPLATES for utility generation.
- ▶ Generate *ad-hoc* DB2 utilities.
- ▶ Migrate DB2 catalog statistics.
- ▶ Various DB2 performance queries related to RTS to help determine if a COPY, REORG, RUNSTATS or a resize of the objects in question might be useful.

## 5.8.2 DB2 Utility Enhancement Tool

The DB2 Utility Enhancement Tool provides additional functionality that is currently not provided in LOAD, REORG, and CHECK DATA. These functions were not added to the DB2 utilities because some of these functions were able to be easily added to this tool to enhance the use of these three utilities.

This product also has some thread management capabilities, such as canceling a thread or preventing access to certain objects temporarily to allow utilities like an online REORG to complete.

## 5.8.3 DB2 Automation Tool

DB2 Automation Tool allows you to set up a process to run a batch job that will generate utility jobs based on specific criteria so that you can REORG those objects where RTS columns signify that a reorganization might be beneficial. The process can run nightly and allow you to set up jobs to eventually be run by a scheduler. This tool helps DBAs by allowing them to not have to worry about running utilities; you set it up and the tool generates the right jobs when needed.

## 5.8.4 DB2 High Performance Unload for z/OS

The main characteristic of the DB2 High Performance Unload is that it allows you to run this utility outside of the DB2 engine. As a result, applications that are running in the same BP are not affected by the utility I/O. The utility generally uses less CPU than the DB2 UNLOAD utility and also allows for many different external formats during the unload process. You can unload data into multiple output files and many more options exist to put the data in the unloaded data in the right format.

## 5.8.5 DB2 Recovery Expert for z/OS

DB2 Recovery Expert is a GUI tool that helps you manage a DB2 recovery scenario. It analyzes the different assets that are available to perform a recovery, say to a point in time, and gives you the relative cost for each scenario. From this list, you can choose the one that is right for the application.

### **5.8.6 DB2 Change Accumulation Tool**

The DB2 Change Accumulation Tool provides a process to quickly recover an object. You take copies of all of the changes that have been made against an object and use the tool to place these changes in a mini-log. This mini-log can be used as input to a recover or create a COPY SHRLEVEL REFERENCE.

### **5.8.7 DB2 Storage Management Utility**

The DB2 Storage Management Utility allows you to view storage information about data sets, including extent information for table spaces and indexes. It also generates storage utilization reports that you can review in order to analyze object sizes and space utilization. An option to resize objects is also useful.

### **5.8.8 Optim Query Tuner for DB2 for z/OS and Optim Query Workload Tuner for DB2 for z/OS**

These tools provide the ability to analyze an SQL statement or an entire workload and give you the necessary recommendations via advisory functions to get the best performance out of these queries with the least amount of effort.

### **5.8.9 DB2 Buffer Pool Analyzer**

DB2 Buffer Pool Analyzer is a tool you can use to evaluate the current performance of the buffer pools and group buffer pools. It makes recommendations for these buffer pools based on sample trace data collected. It allows you to simulate the recommendation before making the changes, in order for you to verify that the changes will result in better performance.



## Sort processing

Many DB2 utilities use DFSORT to sort either the table space or index data. The following utilities use DFSORT:

- ▶ LOAD
- ▶ REORG TABLESPACE and REORG INDEX
- ▶ REBUILD INDEX
- ▶ RUNSTATS
- ▶ CHECK DATA, CHECK INDEX and CHECK LOB

In this chapter, we focus on recent enhancements concerning DB2 and DFSORT. We give recommendations to get the best sorting performance and how to avoid sort failures. We also explain how sort workload can be redirected to a zIIP processor when available. This chapter includes the following topics:

- ▶ The DFSORT program
- ▶ Sort data sets used by DB2 utilities
- ▶ Best DFSORT default options for DB2
- ▶ Main memory used by DFSORT
- ▶ DSNUTILB and DSNUTILS region sizes
- ▶ DFSORT zIIP offload capability
- ▶ Debugging DFSORT problems
- ▶ DB2 allocated sort work data sets

## 6.1 The DFSORT program

The DB2 Utilities Suite is designed to work with the DFSORT program. DB2 utilities can use DFSORT regardless of whether or not you have a license for DFSORT on your system. The DB2 Utilities Suite only uses the SORT and MERGE functions of DFSORT. If you want to use DFSORT for any other uses outside of this limited DB2 support, you must separately order and license DFSORT. Use of DFSORT by DB2 utilities does not interfere with the use of another sort product for other purposes.

The DB2 utilities call DFSORT using the new DFSORT-only aliases of ICEDFSRT and ICEDFSRB instead of ICEMAN and ICEBLDX. There is no interference with another sort product in case DFSORT is not your primary sort product.

Even if you do not have a license for DFSORT, IBM provides central service for DFSORT problems related to the use of DFSORT by DB2 utilities. IBM already provides all DFSORT service (PTFs) to all z/OS customers. If you have any problem when using DFSORT, you should open a PMR against DB2. If necessary, DB2 support will transfer the problem to DFSORT.

You should ensure that you are current on both DFSORT and DB2 maintenance. When in DB2 V8, be sure that PK97713 is applied if you do not have a DFSORT licence.

If DFSORT is installed as your primary z/OS sort product, no additional action is required to make DFSORT available to DB2 when installing DB2 for the first time.

If DFSORT is not installed as your primary z/OS sort product, you do have some additional actions to make DFSORT available to DB2. These actions involve:

- ▶ Installing the DFSORT SVC to generate DFSORT SMF 16 records
- ▶ Making the DFSORT libraries available in the LPA and LINKLIST
- ▶ Changing DFSORT default options for optimal use by DB2
- ▶ Verifying that your ACS routines do not allocate DFSORT data sets to virtual I/O (VIO)
- ▶ Excluding DFSORT data sets from any OEM product that reduces space allocations

For more information about making DFSORT available to DB2, refer to informational APARs II14047 and II14213.

## 6.2 Sort data sets used by DB2 utilities

The DB2 utilities need a lot of data sets for sorting purposes. We first explain and categorize the different types. We can distinguish between:

- ▶ Work data sets used by the DB2 utility outside DFSORT
- ▶ Work data sets used by DFSORT

## 6.2.1 Work data sets used outside DFSORT

These data sets are also called *WORKDDN* data sets. There are two work data sets for sort input and sort output. These data sets have following characteristics:

- ▶ They are specified by the *WORKDDN* option.
  - The default DD name for sort input is *SYSUT1*.
  - The default DD name for sort output is *SORTOUT*.
- ▶ They must be allocated by the user either in the JCL or dynamically allocated via a *TEMPLATE*.
- ▶ The normal disposition for such data sets is (NEW,DELETE,CATLG), because they will be needed during *RESTART* if the utility fails for some reason.

If no space parameters are provided in the template definition, DB2 tries to calculate the right space parameters. This is called intelligent data set sizing.

We suggest to always use templates with intelligent data set sizing for the *WORKDDN* data sets. Use template variables to make their names unique in case parallel utility jobs are running. Do not allocate them in the JCL.

## 6.2.2 Work data sets used by DFSORT

DFSORT uses the following types of data sets:

- ▶ Work data sets used for sorting keys for the indexes (LOAD,REORG,REBUILD)
- ▶ Work data sets used for sorting data (REORG)
- ▶ Work data sets used by RUNSTATS and inline statistics
- ▶ DFSORT message data sets

DFSORT work data sets are identified by their DD names. They have the following general characteristics:

- ▶ They can be allocated by the user in the JCL, they can be dynamically allocated by DFSORT, or they can be dynamically pre-allocated by DB2 itself before invoking DFSORT. To have them dynamically allocated, specify the device type via the *SORTDEVT* keyword.

If you do not specify *SORTDEVT*, the device type specified in the DFSORT *DYNALOC* parameter is used when DFSORT allocates the work data sets or when DB2 pre-allocates them.

- ▶ The normal disposition for such data sets is (NEW,DELETE,DELETE), because they will be allocated NEW during *RESTART* if the utility fails for some reason.
- ▶ Each sort subtask that uses a work data set also requires a message data set where DFSORT puts its messages.

We suggest you always have the sort work data sets allocated dynamically and let DB2 determine the optimal number of data sets (refer to 6.8, “DB2 allocated sort work data sets” on page 159 for details). This optimizes performance of the sort and largely reduces the maintenance of the utility job JCL.

Let us look at the different work data sets used by DFSORT:

- ▶ *SORTWKnn* data sets

Temporary data sets for sort input and output when sorting keys if Parallel Index Build (PIB) is not used. *nn* identifies one or more data sets that are to be used by the sort task for LOAD, REBUILD, and REORG. The *SORTWK01* data set is also used when collecting distribution statistics by RUNSTATS. The used message data set is *UTPRINT*.

► *SWnnWKmm* data sets

Temporary data sets for sort input and output when sorting keys if Parallel Index Build (PIB) is used. nn identifies the subtask pair (one per index), and mm identifies one or more data sets that are to be used by that subtask pair when executing LOAD, REBUILD, or REORG. The used message data set is UTPRINnn.

► *DATAWKnn* data sets

Temporary data sets for sort input and output used by REORG for the data sort. nn identifies one or more data sets that are to be used by the sort task. The used message data set is UTPRINT.

► *DAnnWKmm* data sets

Temporary data sets used by REORG for unload parallelism. nn identifies the subtask pair (one per partition), and mm identifies one or more data sets that are to be used by that subtask pair. The used message data sets are DTPRINnn.

► *STATWK01* data sets

Temporary data sets for sort input and output when collecting distribution statistics for columns groups (RUNSTATS and inline statistics). The used message data set is RNPRIN01.

► *ST01WKnn* data sets

Temporary data sets for sort input and output when collecting frequency statistics on DPSIs or when TABLESPACE TABLE COLGROUP FREQVAL is specified (RUNSTATS and inline statistics). The used message data set is STPRIN01.

► *ST02WKnn*

Temporary data sets for sort input and output when collecting frequency statistics for column groups (RUNSTATS and inline statistics).

► UTPRINT

A data set that contains messages from DFSORT. Required for the LOAD, REBUILD, and REORG utilities.

► UTPRINnn

Data sets that contain messages from DFSORT from utility subtasks pairs. These data sets are dynamically allocated when UTPRINT is allocated to SYSOUT.

► DTPRINnn

Data sets that contain messages from UNLOAD parallelism functions. These data sets are dynamically allocated when UTPRINT is allocated to SYSOUT.

► STPRIN01

A data set that contains messages from DFSORT. This data set is required when frequency statistics are collected on DPSIs or when TABLESPACE TABLE COLGROUP FREQVAL is specified.

► RNPRIN01

A data set that contains messages from DFSORT. This data set is required when distribution statistics are collected for column groups.

► SORTDEVT

Gives DB2 the device type of where to allocate the sortwork data sets. The specification of this keyword allows DB2 to perform dynamic data set allocation, unless the user overrides it with their own DFSORT DYNALLOC parameter. For REORG unload/reload partition parallelism, DB2 requires the specification of SORTDEVT so that the optimal degree of parallelism can be determined.

**Tip:** Try to always specify the SORTDEVT keyword in the utility control statements. This allows for an optimal degree of parallelism.

Sort work data sets cannot span volumes. Smaller volumes require more sort work data sets to sort the same amount of data; therefore, large volume sizes can reduce the number of needed sort work data sets. Since z/OS V1.7, it is possible to define volumes with more than 64,000 tracks. We recommend that at least 1.2 times the amount of data to be sorted be provided in sort work data sets on disk. Using two or three large SORTWKnn data sets are preferable to several small ones.

DFSORT runs most efficiently with a small number of data sets. A small number of data sets also benefits the maximum degree of parallelism for utilities, so you can likely improve the elapsed time, if you have large sort volumes to reduce the number of data sets for very large sorts.

When assigning a data class, it is important that you turn off the ability to allocate sort work data sets on multiple volumes (either directly defining them preventively as multi-volume or letting space constraint relief expand a data set to multiple volumes). DFSORT can only use the part of the sort work data set that resides on the first volume; all other parts on an additional volume will go unused by DFSOR, which wastes disk space and likely causes SORT CAPACITY EXCEEDED failures.

Ensure that sort work allocations are directed to those devices with the fastest available random I/O service times. Spread I/O for sort work files by assigning logical volumes on different physical devices to avoid cache and channel contention. In general, sort work EXCPs are large, which could have a noticeable impact on measured disconnect times for other applications, hence isolate sort work devices particularly from data with critical response time requirements

SORTNUM tells DB2 how many sort work data sets to use for each sort invocation. This is only passed on to DFSORT in the DYNALLOC option statement and tells DFSORT into how many data sets the sort work space should be divided. DB2 allocated sort work data sets do not use this number; they are always allocated as large as possible to result in the smallest possible number of data sets.

DB2 passes the SORTDEVT and SORTNUM values to DFSORT by the DYNALLOC option. DB2 also estimates how many rows are to be sorted and passes this information to DFSORT by the FILSZ option. The values that are passed to DFSORT can be seen in message ICE000I in the corresponding DFSORT message data set, as shown in Example 6-1. In this example, we specified SORTDEVT 3390 SORTNUM 7 and DB2 passes 1248795 rows to sort.

*Example 6-1 Determining the sort parameters passed to DFSORT in the UTPRINnn message data set*

---

```

ICE200I 0 IDENTIFIER FROM CALLING PROGRAM IS 0001
ICE143I 0 BLOCKSET    SORT  TECHNIQUE SELECTED
ICE250I 0 VISIT http://www.ibm.com/storage/dfsor FOR DFSORT PAPERS, EXAMPLES AND MORE
ICE000I 0 - CONTROL STATEMENTS FOR 5694-A01, Z/OS DFSORT V1R10 - 17:11 ON MON OCT 19, 2009 -
      SORT FIELDS=(00007.0,00020.0,A,00001.0,00005.0,A),FORMAT=BI,FILSZ=E0000*
      00001248795,DYNALLOC=(3390,07)
      RECORD TYPE=F,LENGTH=(0026,0026,0026)
      OPTION MSGPRT=ALL,SORTDD=SW01,MSGDDN=UTPRIN01,MAINSIZE=MAX
ICE201I F RECORD TYPE IS F - DATA STARTS IN POSITION 1
ICE751I 0 C5-K48846 C6-K90014 C7-K45047 C8-K46331 E4-BASE    C9-BASE    E5-K48846 E6-BASE
E7-K48846
ICE193I 0 ICEAM2 INVOCATION ENVIRONMENT IN EFFECT - ICEAM2 ENVIRONMENT SELECTED
ICE089I 5 DB2R6X .REBUILD .      , INPUT LRECL = 26, TYPE = F
ICE093I 0 MAIN STORAGE = (MAX,33554432,33554432)
ICE156I 0 MAIN STORAGE ABOVE 16MB = (33497072,33497072)

```

---

When UTSORTAL is set to YES, DB2 uses RTS for the estimates.

When not using the new SORTNUM elimination feature (subsystem parameter UTSORTAL=NO) and no SORTNUM is specified, DB2 queries the DFSORT installation option value DYNALOC (which has a default of 4) to be used for SORTNUM.

When not using the new SORTNUM elimination feature (subsystem parameter UTSORTAL=NO), DB2 uses the RUNSTATS SPACE statistics from the DB2 catalog to estimate the FILSZ value. If RUNSTATS has not been run, when the table space is compressed or the table space contains rows with VARCHAR columns, DB2 might not be able to accurately estimate the number of rows. If the estimated number of rows is too high and the sort work space is not available or if the estimated number of rows is too low, DFSORT might fail and cause an abend.

If after running RUNSTATS the problem still persists, you can also override the DB2 row estimate in FILSZ using control statements that are passed to DFSORT. One way of doing this action is shown Example 6-2, where we override the FILSZ value and set it to 2000000 using a DFSPARM DD statement. The 'U' before the 2000000 is important and tells DFSORT that this is an estimation and to perform file size calculations based on 2000000, while avoiding termination if 2000000 is not exact. If you do not specify 'U', you get the following error message:

```
ICE047A 1 RECORD COUNT OFF, SPECIFIED 2000000, RECEIVED 1248795
```

The utility fails. This failure can be prevented by setting the DFSORT installation option FSZEST=YES.

*Example 6-2 Overriding the DB2 estimated FILSZ value*

---

```

//DFSPARM DD *
      OPTION FILSZ=U2000000
/*

```

---



To verify that the new FILSZ is used, check the ICE253 message (new in DFSORT V1R10), as shown in Example 6-3.

*Example 6-3 Verifying if the new FILSZ was used*

---

```
ICE253I 0 RECORDS SORTED - PROCESSED: 1248795, EXPECTED: 2000000
```

---

However, using control statements to override FILSZ estimates that are passed to DFSORT applies for all invocations of DFSORT in the job step, including any sorts that are done in any other utility that is executed in the same step. The result might be reduced sort efficiency or an abend due to an out-of-space condition. Also, this DFSPARM FILSZ override does not have an effect when using DB2 allocated sort work data sets. DB2 will in that case still allocate the sort work data sets based on its own estimates. So use this override in special situations. For UTSORTAL=YES, you could set a changed value in TOTALROWS or TOTALENTRIES if these are not correct and cause sort failures.

The use of Virtual I/O (VIO) for DFSORT work data sets is *not* recommended. VIO=NO is DFSORT's shipped installation default and the recommended setting for the VIO parameter. If your site's ACS routines allocate DFSORT work data sets to VIO, we recommend that you change your ACS routines to avoid using VIO for SORTWK\*, STATWK\*, DATAWK\*, DAnnWK\*, STnnWK\*, and SWnnWK\* data definitions. This is because DFSORT already performs storage hierarchy management (the efficient balancing of processor memory and disk storage), and mixing this with VIO, which uses processor memory, results in suboptimal efficiency.

When dynamically allocating a work data set, DFSORT uses a primary allocation of 0 tracks for the initial allocation of work data sets. This can impact the decisions made by DFSMS Automatic Class Selection (ACS) routines if the &size variable is being used to assign a unit (such as VIO) or select a storage class and group based on data set size. We recommend that the &maxsize variable be used as a basis for decisions related to data set size. Because DFSORT work data set allocations contain a secondary space quantity, the &maxsize variable allows for a more accurate decision of the assignment of unit, storage class, storage group, and so on. While DFSORT performs the initial work data set allocations with a primary space quantity of 0, it attempts to extend them once the accurate amount of required space is determined.

Should a work data set, dynamically allocated by DFSORT, reside on a volume that does not have sufficient space available, DFSORT invokes **recovery routines** that may result in the data set being deleted and re-allocated. However, this is not the case if the work data sets are preallocated by DB2.

Exclude the SORTWK\*, STATWK\*, DATAWK\*, DAnnWK\*, STnnWK\*, and SWnnWK\* data sets from any OEM products that reduce space allocations while running concurrent utilities. If the percent free of your sort pool is low, you can either reduce the number of concurrently running utilities or you can increase the number of volumes in your sort pool. If you increase the number of volumes, you can then monitor utilization and bring that number back down if its consistently underutilized. As a general rule, your sort pool should average over 20% free space to allow for optimal allocation of DFSORT work data sets and reduce the risk of failures due to lack of work space (ICE083A, ICE046A, and so on)

## 6.3 Best DFSORT default options for DB2

DFSORT exploits processor storage through the use of either Hiperspaces, Dataspaces, or Memory Objects. DFSORT exploits the System z 64-bit real architecture by backing storage and data spaces in real storage above 2 GB, and by using main storage instead of expanded storage for Hipersorting.

It can be useful to check the DFSORT installed default options when running DB2 utilities. The default values can be listed with a job like the one listed in Example 6-4.

*Example 6-4 List DFSORT installation defaults*

---

```
//ICEDFLT EXEC PGM=ICETOOL
//TOOLMSG DD SYSOUT=*
//DFSMSG DD SYSOUT=*
//LIST1 DD SYSOUT=*
//TOOLIN DD *
  DEFAULTS LIST(LIST1)
```

---

The output of this job is shown in Example 6-5.

*Example 6-5 DFSORT installation defaults*

---

z/OS DFSORT V1R10 MERGED PARMLIB/ICEMAC DEFAULTS - 1 -

\* IBM-SUPPLIED DEFAULT (ONLY SHOWN IF DIFFERENT FROM THE SPECIFIED DEFAULT)

ITEM	JCL (ICEAM1) VALUE	INV (ICEAM2) VALUE	TSO (ICEAM3) VALUE
-----	-----	-----	-----
ENABLE	NONE	NONE	NONE
ABCODE	MSG	MSG	MSG
ALTSEQ	SEE BELOW	SEE BELOW	SEE BELOW
ARESALL	0	0	0
AREINV	NOT APPLICABLE	0	NOT APPLICABLE
CFW	YES	YES	YES
CHALT	NO	NO	NO
CHECK	YES	YES	YES
CINV	YES	YES	YES
COBEXIT	COB2	COB2	COB2
DIAGSIM	NO	NO	NO
DSA	64	64	64
DSPSIZE	MAX	MAX	MAX
DYNALOC	(SYSDA,4)	(SYSALLDA,4)	(SYSDA,4)
DYNSPC	256	8192	256
		* 256	
EFS	NONE	NONE	NONE

---

ICETOOL lists the DFSORT defaults for four invocation environments: ICEAM1, ICEAM2, ICEAM3, and ICEAM4. DB2 utilities are only affected by the ICEAM2 (INV) environment for program invoked sorts.

When reviewing the DFSORT installation defaults, pay particular attention to the following parameters:

► **EXPMAX**

This parameter can be used to limit the amount of storage DFSORT considers available for Hiperspace™ and Memory Object sorting even when resources are available. This should be set to the shipped default of MAX to allow DFSORT to take full advantage of available resources.

► **EXPOLD**

This parameter controls the amount of “old frames” that DFSORT considers available for Hiperspace and Memory Object sorting. In large main storage environments, where there are address spaces with large working sets that have not been recently referenced, the shipped default of MAX can lead to auxiliary storage shortages. To prevent this from happening, we recommend setting EXPOLD=0. This causes DFSORT to only consider available frames in its calculation of available resources for Hiperspace and Memory Object sorting.

► **MOSIZE and HIPRMAX**

These parameters can limit the amount of Hiperspace or Memory Object storage that can be allocated by each individual sort. We recommend these parameters be left at their shipped defaults of MOSIZE=MAX and HIPRMAX= OPTIMAL. It is better to control memory object and Hiperspace usage with the global controls of EXPMAX and EXPOLD.

► **DYNALOC**

This parameter specifies the unit name and default number of sortwork data sets each sort will allocate. The impact of this parameter is minimized if running with the SORTNUM elimination feature or when using SORTNUM in the control statements. However, for sorts taking the system default, make sure this is set properly for the environment. Avoid setting this to extremely high defaults, such as 128 or 255, because this will limit parallelism. Usually, a default of 8 or 16 works well, and then you can use SORTNUM to override for the really large sorts.

► **DSA**

This parameter sets the limit for DFSORT's Dynamic Storage Adjustment feature. This means that if you specify a DSA value greater than the TMAXLIM value, you allow DFSORT to use more storage than the TMAXLIM value if doing so would improve performance. DFSORT only tries to obtain as much storage as needed to improve performance up to the DSA value. The utilities use the DSA installation default as a limit. In an environment where utilities will be executing very large sorts (greater than 32 GB), increase this parameter from its shipped default of 64 MB to 128 MB.

► **SIZE**

This parameter specifies the maximum amount of main storage DFSORT may use. The normal value is MAX, which means that DFSORT will determine the amount of storage available and use that value.

► **MAXLIM and TMAXLIM**

These parameters specify the limits to the amount of maximum storage available. TMAXLIM is the limit for the total storage above and below 16 MB virtual, when SIZE=MAX is in effect. MAXLIM is the limit for the storage below 16 MB virtual. Recommended values are MAXLIM=1 MB and TMAXLIM=6 MB.

► SMF

This parameter specifies the level of SMF recording. The shipped default is SMF=NO. Changing this to SMF=FULL will instruct each sort to generate an SMF record type 16. These records are very useful in analyzing DFSORT performance, if you have the required skills.

Starting from DFSORT V1R5, DFSORT exploits main storage through the use of Memory Objects above the 2 GB bar. Memory Object sorting will only be used if MEMLIMIT is non-zero. If MEMLIMIT is not set in your JCL, then it defaults to the SMF MEMLIMIT value specified in the SMFPRMxx member of SYS1.PARMLIB. If MEMLIMIT is not specified in SMFPRMxx, then that default is zero. Specifying REGION=0M always implies MEMLIMIT=NOLIMIT. We recommend not using a MEMLIMIT parameter in your JCL and use the default MEMLIMIT setting for most of your utility jobs.

Note that the recommended value of EXPOLD=0 may have a negative impact on sort elapsed times because it will limit the resources DFSORT considers available for Hiperspace and Memory Object sorting. If you are sure that your paging subsystem is large enough to support an increased amount of page outs from main storage, you might consider increasing this value to allow page stealing from other address spaces. Use RMF to monitor your page data set usage and paging activity when increasing EXPOLD.

DFSORT dynamically chooses between using data space sorting, memory object sorting, and Hipersorting; DFSORT selects the one that provides the best performance for the particular sort. However, for DB2 utilities, data space sorting will be rare because data space sorting is usually more expensive. The sort method used can easily be determined from the UTPRINnn message data sets shown in Example 6-6. Here Hipersort was used. These messages also show the storage used by DFSORT above the 2 GB bar.

*Example 6-6 Determining the sort method used from the DFSORT message data sets*

---

```
ICE199I 0 MEMORY OBJECT STORAGE USED = 0M BYTES
ICE180I 0 HIPERSPACE STORAGE USED = 44100K BYTES
ICE188I 0 DATA SPACE STORAGE USED = 0K BYTES
```

---

Generally speaking, the object memory sorting will use more CPU than Hipersorting, but object memory sorting is eligible for zIIP offload. Refer to 6.6, “DFSORT zIIP offload capability” on page 155 for more information. We suggest enabling all of these sort methods and always let DFSORT choose the best sort method (with the default DFSORT options, all are enabled).

For more information about DFSORT and 64 bit architecture, refer to APAR II13495.

In DFSORT V1R10, it is possible to dynamically activate changed DFSORT options via concatenated PARMLIB ICEPRMxx members:

- Can be activated at any time with START ICEOPT from console
- Merged with ICEMAC values and defaults at run time
- Up to 10 members can be specified on START ICEOPT command

Let us demonstrate how we dynamically changed the DSA option from 64 MB to 128 MB for DB2 utilities. We first created a member ICEPRM02, as shown in Example 6-7, in the MVS PARMLIB and then executed the console command START ICEOPT,ICEPRM=02.

*Example 6-7 DFSORT ICEPRMxx parmlib member*

---

```
INV
    DSA=128
```

---

The output of the command is shown in Example 6-8.

*Example 6-8 Dynamically changing a DFSORT default option*

---

```

SDSF OPERLOG  DATE 10/29/2009      2 WTORS          COMMAND ISSUED
COMMAND INPUT ==> /START ICEOPT,ICEPRM=02          SCROLL ==> 7
RESPONSE=SC63      ICEP500I ICEOPT: COMPLETED - OPTIONS CLEARED/SAVED IN AREA 2

```

---

If we list the defaults afterwards, we can see that the INV value for DSA is changed from the initial default of 64 MB to 128 MB, as shown in Example 6-9.

*Example 6-9 List of the changed DFSORT parameters*

ITEM	JCL (ICEAM1) VALUE	INV (ICEAM2) VALUE	TSO (ICEAM3) VALUE
-----	-----	-----	-----
ENABLE	NONE	NONE	NONE
ABCODE	MSG	MSG	MSG
ALTSEQ	SEE BELOW	SEE BELOW	SEE BELOW
ARESALL	0	0	0
ARESINV	NOT APPLICABLE	0	NOT APPLICABLE
CFW	YES	YES	YES
CHALT	NO	NO	NO
CHECK	YES	YES	YES
CINV	YES	YES	YES
COBEXIT	COB2	COB2	COB2
DIAGSIM	NO	NO	NO
DSA	64	128 * 64	64
DSPSIZE	MAX	MAX	MAX

---

## 6.4 Main memory used by DFSORT

With main memory, we refer to virtual storage used under the 2 GB bar. Before DB2 utilities invoke DFSORT they will query the DFSORT installation parameters. The parameters queried are SIZE, DSA, TMAXLIM, and in some cases, MAXLIM. DB2 will also determine the available amount of virtual storage for the utility job, under the 2 GB bar, based on the region parameter in the JCL. Because DB2 utilities calculate the number of parallel tasks based on the amount of virtual storage available, they need to control the amount of virtual storage each sort will allocate. They do that by passing the MAINSIZE parameter to each individual sort task specifying the exact amount of virtual storage to use.

In a simplified way, we can say that in most cases the MAINSIZE parameter is calculated by dividing the available storage for the job by the number of parallel tasks. This amount is then decreased to reserve some memory for the utility itself. If this result is larger than DSA or TMAXLIM, it is limited to the highest value of DSA and TMAXLIM. For example, if we have DSA=64 MB, TMAXLIM=6 MB, an available storage of 100 MB and four sort tasks, DB2 will chose a MAINSIZE parameter value around 23 MB. If the available storage is 300 MB, the resulting MAINSIZE will be 64 MB, because 73 MB is bigger than the DSA limit of 64 MB. This calculated value of MAINSIZE is then passed to each individual sort task and is equal for each sort task. However, for some sort tasks, this value may be bigger then the actual required because not all sort tasks require the same amount of storage. Also, we do not use the Dynamic Storage Adjustment (DSA) feature, which allows DFORT to use more storage if doing so would improve performance.

**Attention:** Since DFSORT APAR PK01155 (2005), it is no longer possible to overwrite the MAINSIZE, DSA, TMAXLIM and DYNALOC=N parameters passed from DB2 to DFSORT by a DFSPARM statement, but you can still override the device type of the DYNALOC parameter.

Therefore, PK79136 (DB2 9 only) changes the way DB2 tells which amount of virtual storage to use for individual sort tasks. With PK79136, DB2 now allows DFSORT to use the DSA feature. It always specifies MAINSIZE=MAX on the invocation of DFSORT and limits the storage consumption of each sort task by specifying an individual DSA size based on available storage and the DSA installation option setting. As a result, the sort tasks will only use the storage they actually need between TMAXLIM (initial value) and the passed DSA.

This situation can also be seen in the DFSORT message data sets UTPRINnn, as shown in Example 6-10. This is an example on a V8 system without the new support. The calculated and passed MAINSIZE is 11826 KB. The total main storage used is 12109824 bytes.

*Example 6-10 Determining the storage used by an individual sort task before PK79136*

---

```
ICE143I 0 BLOCKSET    SORT  TECHNIQUE SELECTED
ICE250I 0 VISIT http://www.ibm.com/storage/dfsor FOR DFSORT PAPERS, EXAMPLES AND MORE
ICE000I 0 - CONTROL STATEMENTS FOR 5694-A01, Z/OS DFSORT V1R5 - 22:58 ON MON OCT 19, 2009 -
          SORT FIELDS=(00007.0,00007.0,A,00001.0,00005.0,A),FORMAT=BI,FILSZ=U0000*
          00000000391,DYNALOC=(SYSDA,07)
          RECORD TYPE=F,LENGTH=(0013,0013,0013)
          OPTION MSGPRT=ALL,SORTDD=SW03,MSGDDN=UTPRIN03,MAINSIZE=011826K
ICE201I F RECORD TYPE IS F - DATA STARTS IN POSITION 1
ICE751I 0 C5-K90013 C6-K90013 C7-K90000 C8-K42135 E4-K90007 C9-BASE  E5-K44563 E6-K34782 E7-K44563
ICE193I 0 ICEAM2 ENVIRONMENT IN EFFECT - ICEAM2 INSTALLATION MODULE SELECTED
ICE089I 1 SIDDAGOA.STEP01 .      , INPUT LRECL = 13, TYPE = F
ICE092I 0 MAIN STORAGE = (12109824,12109824,12109824)
ICE156I 0 MAIN STORAGE ABOVE 16MB = (11980939,11980939)
ICE127I 0 OPTIONS: OVFL0=RC0 ,PAD=RC0 ,TRUNC=RC0
          ,SPANINC=RC16,VLSCMP=N,SZERO=Y,RESET=Y,VSAMEMT=Y,DYNSPC=256
ICE128I 0 OPTIONS: SIZE=12109824,MAXLIM=1048576,MINLIM=450560,EQUALS=N,LIST=Y,ERET=RC16,MSGDDN=UTPRIN03
ICE129I 0 OPTIONS: VIO=N,RESDNT=ALL ,SMF=NO  ,WRKSEC=Y,OUTSEC=Y,VERIFY=N,CHALT=N,DYNALOC=(SYSDA
          ,007),ABCODE=MSG
ICE130I 0 OPTIONS: RESALL=0,RESINV=0,SVC=109 ,CHECK=Y,WRKREL=Y,OUTREL=Y,CKPT=N,STIMER=Y,COBEXIT=COB2
ICE131I 0 OPTIONS: TMAXLIM=6291456,ARESALL=0,ARESINV=0,OVERRGN=16384,CINV=Y,CFW=N,DSA=0
ICE132I 0 OPTIONS: VLSHRT=N,ZDPRINT=Y,IEXIT=N,TEXT=N,LISTX=N,EFS=NONE  ,EXITCK=S,PARMDDN=DFSPARM
          ,FSZEST=N
ICE133I 0 OPTIONS: HIPRMAX=0      ,DSPSIZE=0  ,ODMAXBF=0,SOLRF=Y,VLLONG=N,VSAMIO=N,MOSIZE=MAX
ICE235I 0 OPTIONS: NULLOUT=RC0
```

---

In Example 6-11, we show the DFSORT message output on a DB2 9 system with PK79136 applied. The passed MAINSIZE is MAX and the total storage available is 6291456 bytes, which is identical to the initial TMAXLIM setting. This means that the initial TMAXLIM is enough to do the sort. The DSA value that is calculated by DB2 and passed to DFSORT as the upper virtual storage to use is 19 MB, but this value is not shown in this report. In case the sort decides to use more than the initial TMAXLIM value, this would be reflected in the IEC093 message and in a non-zero DSA value in ICE131.

*Example 6-11 Determining the storage used by an individual sort task with PK79136/PK92248*

---

```

ICE200I 0 IDENTIFIER FROM CALLING PROGRAM IS 0002
ICE805I 0 JOBNAME: DB2R6Y , STEPNAME: REORG
ICE802I 0 BLOCKSET TECHNIQUE IN CONTROL
ICE201I F RECORD TYPE IS F - DATA STARTS IN POSITION 1
ICE992I 0 RA 0 WR 0 TR 0
ICE751I 0 C5-K48846 C6-K90014 C7-K45047 C8-K46331 E4-BASE C9-BASE E5-K48846 E7-K48846
ICE143I 0 BLOCKSET SORT TECHNIQUE SELECTED
ICE250I 0 VISIT http://www.ibm.com/storage/dfsor/papers\_examples\_and\_more
ICE000I 0 - CONTROL STATEMENTS FOR 5694-A01, Z/OS DFSORT V1R10 - 14:07 ON THU OCT 22, 2009 -
        SORT FIELDS=(00007.0,00029.0,A,00001.0,00005.0,A),FORMAT=BI,FILSZ=U0000*
        00001248795
        RECORD TYPE=F,LENGTH=(0035,0035,0035)
        OPTION MSGPRT=ALL,SORTDD=SW02,MSGDDN=UTPRIN02,MAINSIZE=MAX,USEWKDD
ICE201I F RECORD TYPE IS F - DATA STARTS IN POSITION 1
ICE992I 0 RA 0 WR 0 TR 0
ICE751I 0 C5-K48846 C6-K90014 C7-K45047 C8-K46331 E4-BASE C9-BASE E5-K48846 E7-K48846
ICE193I 0 ICEAM2 INVOCATION ENVIRONMENT IN EFFECT - ICEAM2 ENVIRONMENT SELECTED
ICE089I 1 DB2R6Y .REORG . , INPUT LRECL = 35, TYPE = F
ICE093I 0 MAIN STORAGE = (MAX,6291456,6291456)
ICE156I 0 MAIN STORAGE ABOVE 16MB = (6234096,6234096)
ICE127I 0 OPTIONS: OVFL0=RC0,PAD=RC0,TRUNC=RC0,SPANINC=RC16,VLSCLP=N,SZERO=Y,RESET=Y,VSAMEMT=Y,DYNMPC=8192
ICE128I 0 OPTIONS: SIZE=6291456,MAXLIM=1048576,MINLIM=450560,EQUALS=N,LIST=Y,ERET=RC16,MSGDDN=UTPRIN02
ICE129I 0 OPTIONS: VIO=N,RESNNT=ALL,SMF=NO,WRKSEC=Y,OUTSEC=Y,VERIFY=N,CHALT=N,DYNALOC=N,ABCODE=MSG
ICE130I 0 OPTIONS: RESALL=4096,RESINV=0,SVC=109,CHECK=Y,WRKREL=Y,OUTREL=Y,CKPT=N,COBEXIT=COB2
ICE131I 0 OPTIONS: TMAXLIM=6291456,ARESALL=0,ARESINV=0,OVERRGN=16384,CINV=Y,CFW=Y,DSA=0
ICE132I 0 OPTIONS: VLSHRT=N,ZDPRINT=Y,IEXIT=N,TEXIT=N,LISTX=N,EFS=NONE,EXITCK=S,PARMDDN=DFSPARM,FSZEST=N
ICE133I 0 OPTIONS: HIPRMAX=OPTIMAL,DSPSIZE=MAX,ODMAXBF=0,SOLRF=Y,VLLONG=N,VSAMIO=N,MOSIZE=MAX
ICE235I 0 OPTIONS: NULLOUT=RC0

```

---

PK92248 will limit the initial TMAXLIM value to 6 MB, even if a much higher TMAXLIM value is specified in the DFSORT defaults. When using the Dynamic Storage Adjustment feature in DFSORT, it can only adjust the storage consumption effectively when initial storage, as defined in TMAXLIM, is set to a small value. PK92248 supersedes PK79136, which is flagged in error and should be applied if you do not have it on.

So we can say that with DSA controlled memory consumption, a larger REGION size can be specified for all utility jobs, because small sorts will not exploit the region limit, and large sorts will be able to run more efficiently, as shown in Figure 6-1.

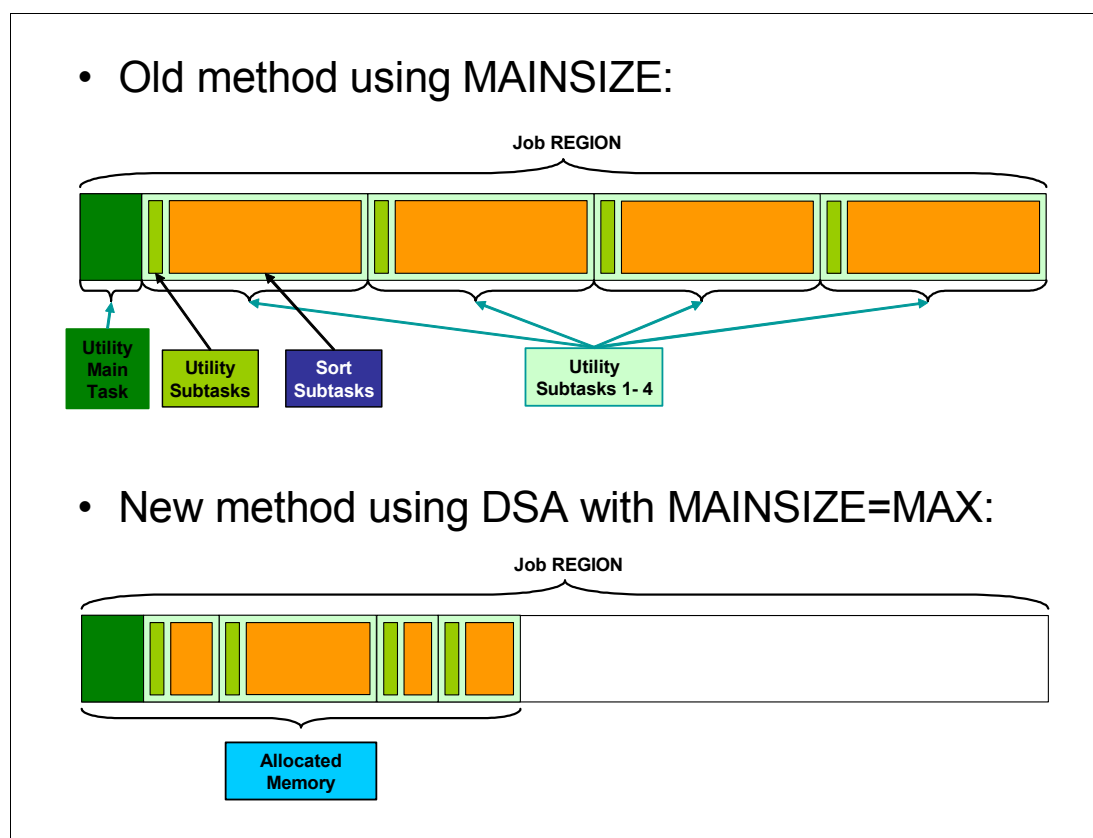


Figure 6-1 Comparing memory limit methods

## 6.5 DSNUTILB and DSNUTILS region sizes

DFSORT may abend or process inefficiently for large sorts when there is not enough main memory available, which can occur especially when the utility runs many sorts in parallel. For large sorts when main memory is constrained, DFSORT will compensate with intermediate merges that increases the need for sort work data sets. One method to avoid this is to ensure that there is sufficient main memory available for large sorts. The information given in Table 6-1 gives a rough estimate on the amount of memory that DFSORT would need to run efficiently.

Table 6-1 Estimated main memory use per DFSORT task

Number of bytes to be sorted	Estimated main memory
1 GB	10 MB
10 GB	30 MB
100 GB	70 MB

The number of bytes to be sorted is equal to record count \* record length.



Increasing the region size of the batch utility job (DSNUTILB) or WLM address space (DSNUTILS) to accommodate the main memory needed for the parallel DFSORT tasks can allow the sorts to succeed. If region=0M is coded on the JOBCARD, then the subtasks can have as much virtual storage as is required, but this can cause real storage exhaustion and excessive paging, especially if multiple utility jobs are run in parallel. Refer to 14.1.1, “Preparing for utility execution” on page 376 for more details about the region parameter. As explained in 6.4, “Main memory used by DFSORT” on page 151, when using DB2 9 with PK79136 and PK92248 applied, larger REGION sizes can be specified than before.

Our recommendation is to always have the sort work data sets allocated dynamically and let DB2 determine the optimal number of data sets (refer to 6.8, “DB2 allocated sort work data sets” on page 159 for more details). This will optimize performance of the sort and largely reduce the maintenance of the utility job JCL.

Be sure to allocate the UTPRINT data set to SYSOUT \*, as shown in Example 6-12. This will allow DFSORT to allocate additional UTPRINxx and DTPRINxx data sets per parallel sort task. If you allocate UTPRINT or UTPRINxx or DTPRINxx to other than SYSOUT=\*, no additional parallel sort tasks will be initiated.

*Example 6-12 Sample JCL for invoking DSNUTILB*

---

```
//RUN EXEC PGM=DSNUTILB,PARM='DB9A,DAVY00'  
//STEPLIB DD DSN=DB9A9.SDSNEXIT,DISP=SHR  
//          DD DSN=DB9A9.SDSNLOAD,DISP=SHR  
//SYSPRINT DD SYSOUT=*  
//UTPRINT DD SYSOUT=*  
//SORTDIAG DD DUMMY  
//SYSIN DD *  
utility input statements  
/*
```

---

The number of parallel DFSORT tasks for utilities may be limited by allocating the UTPRINxx data sets to something other than SYSOUT \*. For example, if UTPRIN01, UTPRIN02, and UTPRIN03 are allocated to real data sets, then at most three parallel sort tasks would be initiated by the utility.

## 6.6 DFSORT zIIP offload capability

In each version of DB2, IBM tries to reduce the CPU consumption of the DB2 utilities to compensate the growth of data of the customers. One example is the DB2 9 CPU reduction during the utility index processing (up to 50%). Another one is the zIIP offload of index maintenance processing to zIIP processors when available in DB2 V8 (also up to 50%).

Sort processing can consume up to 60% of the CPU consumption of a DB2 utility. DB2 APAR PK85889 (DB2 V8 and DB2 9), in conjunction with DFSORT APAR PK85856, introduce a new zIIP offload capability when sorting records in memory, which will be beneficial for most DB2 utilities when using DFSORT. It is only invoked in z/OS 1.10 or higher:

- ▶ With DFSORT V1R10
- ▶ Available for fixed length record sorts (all utility sorts are fixed length except the REORG data sort)
- ▶ Activated when DFSORT selects the object memory sorting method (refer to 6.3, “Best DFSORT default options for DB2” on page 148)

- Indicated by new DFSORT message “ICE256I DFSORT CODE IS ELIGIBLE TO USE ZIIP FOR THIS DB2 UTILITY RUN”

First measurements in IBM Labs indicate up to 50% offload of DFSORT CPU time, which can account for up to 60% of utility CPU time. Results vary by utility, number, and size of indexes, and zIIP use by other applications. Use SMF 30 records for accounting and measurement purposes.

## 6.7 Debugging DFSORT problems

If you have a any problem with DFSORT (for example, you receive DFSORT messages ICE039A or ICE046A or an abend) and the problem can be reproduced, add a DFSPARM DD statement to the failing job and rerun it as shown in Example 6-13. This will allow DFSORT to display additional diagnostic (ICE8xxx and ICE9xxx) messages, and ABEND if a DFSORT error message is issued. Eventually, send the complete JOBLOG and dump (if an error occurs) to the appropriate IBM service representative.

*Example 6-13 Get additional diagnostic messages out of DFSORT*

---

```
//SORTDIAG DD DUMMY
//DFSPARM DD *
          DEBUG ABEND
/*
```

---

We recommend always putting //SORTDIAG DD DUMMY in your utility JCL to have more messages written in the DFSORT message data sets UTPRINT, UTPRINnn, and DTPRINnn.

Recommended DFSORT maintenance includes the following APARs:

- PK25047: ICE046A SORT CAPACITY EXCEEDED can result when DFSORT uses multiple Hipspace and more than 2 GB of Hipspace is required to successfully complete the sort.
- DFSORT APAR PK63409: ICE046A SORT CAPACITY EXCEEDED when the FILSZ estimate is slightly below the actual value. A fix for DFSORT in z/OS 1.10 is provided in PK66899.
- DFSORT APAR PK89889: Avoid a loop when using DB2 PK79136.
- Refer to APAR II14502 for DB2 recommended maintenance on SQL sort and Utility sort.

When a DB2 utility job encounters DFSORT problems, it will put some messages in the SYSPRINT output data set and then abend with S04E RC00E40347, as shown in Example 6-14. The reason code 00E4005 means that one or more sort subtasks have been abending.

*Example 6-14 DB2 utility abending because of DFSORT problems*

---

```
DSNU3340I 294 17:00:39.48 DSNURPIB - UTILITY PERFORMS DYNAMIC ALLOCATION OF SORT DISK SPACE
DSNU3342I 294 17:00:39.91 DSNURPIB - NUMBER OF OPTIMAL SORT TASKS = 5, NUMBER OF ACTIVE SORT TASKS = 5
DSNU395I 294 17:00:39.91 DSNURPIB - INDEXES WILL BE BUILT IN PARALLEL, NUMBER OF TASKS = 15
DSNU044I 294 17:00:40.08 DSNUGSRP - ERROR FROM SORT COMPONENT RC=16, UTILITY STOPPED
DSNU044I 294 17:00:40.09 DSNUGSRP - ERROR FROM SORT COMPONENT RC=16, UTILITY STOPPED
DSNU016I 294 17:00:40.12 DSNUGSAT - UTILITY BATCH MEMORY EXECUTION ABENDED, REASON=X'00E40005'
DSNU016I 294 17:00:40.12 DSNUGSAT - UTILITY BATCH MEMORY EXECUTION ABENDED, REASON=X'00E40005'
DSNU017I 294 17:00:42.33 DSNUGBAC - UTILITY DATA BASE SERVICES MEMORY EXECUTION ABENDED,
REASON=X'00E40347'
```

---

More information about the sort problem can then be found in the individual DFSORT message output data sets UTPRINT and UTPRINnn/DTPRINnn. When you see:

ICE185A 0 AN S13E ABEND WAS ISSUED BY DFSORT, ANOTHER PROGRAM OR AN EXIT (PHASE S 1)

this means that this error was a consequence of another error in another sort subtask.

Typical error messages that will appear in an abending sort subtask are:

- ▶ ICE039A INSUFFICIENT MAIN STORAGE
- ▶ ICE046A SORT CAPACITY EXCEEDED
- ▶ ICE083A RESOURCES WERE UNAVAILABLE FOR DYNAMIC ALLOCATION OF WORK DATA SETS

Typical actions to do when you encounter DFSORT problems are:

- ▶ Make sure VIO=NO (the DFSORT's installation default).
- ▶ Increase the region size of the job to avoid insufficient virtual storage problems.
- ▶ Ensure that there is enough free space available in the sort pool. Eventually, add one or more DASD volumes to it. Ensure that the work data sets are not allocated as multivolume data sets. DFSORT can only use the part of the sort work data set that resides on the first volume.
- ▶ Use the SORTNUM elimination feature and let DB2 dynamically preallocate sort work data sets by:
  - Deleting all the sort related DD cards (SORTWKnn, SW01WKnn, DATAWKnn, DA01WKnn, STATWKnn, and ST01WKnn,UTPRINnn,DTPRINnn) from the JCL., except UTPRINT and SORTDIAG
  - Setting the DB2 subsystem parameter UTSORTAL=YES
  - Removing the SORTNUM parameter from the utility control statement

- Compare the FILSZ value and average record length passed by DB2 with the actual number of records to be sorted and the actual average record length. See Example 6-15. Large discrepancies between the actual values and the information provided by DB2 can cause problems. Eventually, override the FILSZ value with your own value, as shown in Example 6-2 on page 146.

*Example 6-15 Verifying the FILSZ and AVGREC value passed by DB2 with the actual number of rows to be sorted*

---

```

ICE200I 0 IDENTIFIER FROM CALLING PROGRAM IS 0000
ICE805I 0 JOBNAME: DB2R6Y , STEPNAME: REORG
ICE802I 0 BLOCKSET      TECHNIQUE IN CONTROL
ICE201I F RECORD TYPE IS V - DATA STARTS IN POSITION 5
ICE992I 0 RA 0 WR 0 TR 0
ICE751I 0 C5-K48846 C6-K90014 C7-K45047 C8-K46331 E4-BASE   C9-BASE   E5-K48846 B0-K38900 E7-K48846
ICE143I 0 BLOCKSET      SORT TECHNIQUE SELECTED
ICE250I 0 VISIT http://www.ibm.com/storage/dfsor FOR DFSORT PAPERS, EXAMPLES AND MORE
ICE000I 0 - CONTROL STATEMENTS FOR 5694-A01, Z/OS DFSORT V1R10 - 15:07 ON THU OCT 22, 2009 -
          SORT FIELDS=(00030.0,00002.0,A,00005.0,00018.0,A),FORMAT=BI,FILSZ=E0000*
          00001248795
          RECORD TYPE=V,LENGTH=(00189,00189,00189)
          OPTION MSGPRT=ALL,AVGRLN=00189,MAINSIZE=MAX,MSGDDN=UTPRINT,SORTDD=DATA*
          ,USEWKDD
ICE201I F RECORD TYPE IS V - DATA STARTS IN POSITION 5
.....
.....
ICE080I 0 IN MAIN STORAGE SORT
ICE055I 0 INSERT 1248795, DELETE 1248795
ICE054I 0 RECORDS - IN: 0, OUT: 0
ICE134I 0 NUMBER OF BYTES SORTED: 236022255
ICE253I 0 RECORDS SORTED - PROCESSED: 1248795, EXPECTED: 1248795
ICE098I 0 AVERAGE RECORD LENGTH - PROCESSED: 189, EXPECTED: 189
ICE165I 0 TOTAL WORK DATA SET TRACKS ALLOCATED: 5010 , TRACKS USED: 0
ICE199I 0 MEMORY OBJECT STORAGE USED = 278M BYTES
ICE180I 0 HIPERSPACE STORAGE USED = 0K BYTES
ICE188I 0 DATA SPACE STORAGE USED = 0K BYTES

```

---

**Attention:** ICE253I and ICE098I are new in DFSORT V1R10. For older versions of DFSORT, use ICE055I and ICE098I.

When you receive ICE247I INTERMEDIATE MERGE ENTERED, PERFORMANCE MAY BE DEGRADED or multiple messages ICE894I for a single sort, virtual storage is constrained and a intermediate merge has occurred. Increase the amount of virtual storage in that case.

## 6.8 DB2 allocated sort work data sets

Before this new function, DB2 allocated sort work data sets, also called SORTNUM elimination, was introduced to DB2 utilities, specification of the SORTNUM parameter was required when using dynamic allocation of sort work data sets in DFSORT.

Determining the optimal SORTNUM value is not a trivial task. If too few are allocated, the sort could fail due to the inability to allocate the required work space. If too many are allocated, performance can be impacted because the degree of parallelism may be limited due to resource constraints. Note that the number of sort work data sets does not change the total space allocated by DFSORT's dynamic allocation; it merely changes the number and size of the individual sort work data sets. Because each work data set requires virtual storage, an increase in the number of work data sets can reduce the number of tasks DB2 utilities can execute in parallel. Each data set that is allocated will occupy storage below the line, as well as each task that is started. These requirements are taken into consideration for the number of parallel tasks. This is also the reason why DB2 does not let DFSORT allocate as many data sets as it requires to satisfy the estimated sort work space because the number of data sets actually used could exceed the expected number of data sets and the expected consumption of virtual storage that could cause the utility job to fail. If the number of data sets was specified on the DFSORT invocation, the individual data sets may be larger than any volume that is available in the sort pool. In constrained environments, DFSORT might not be able to allocate required disk space in the given number of data sets causing failure of this utility job.

Another problem of the SORTNUM parameter is that it will be used for all DFSORT invocations of a utility and especially REORG TABLESPACE, which has multiple sorts with very different sort characteristics. SORTNUM has to be chosen to accommodate the largest sort of the utility (usually the data sort in REORG). However, the smaller sorts will then also be allocated with the same number of data sets wasting valuable storage below the line and as a result limit the number of parallel tasks. So, if SORTNUM is picked manually for utility jobs, it should be set as small as possible to successfully run the work.

The DB2 utilities SORTNUM elimination feature is designed to let DB2 determine the optimal number of work data sets to allocate. This function was first made available with PK45916 / UK33692 for DB2 Version 8 and PK41899 / UK33636 for DB2 Version 9. Refer to APAR II14296 for more information.

These utilities add new functionality to preallocate sort work space before invoking DFSORT and to enhance the record estimation with the use of real-time statistics (RTS). Preallocation of sort work space by a DB2 utility leaves it with full control over the used resources, so the utility can calculate the best possible degree of parallelism for the currently allocated sort work data sets before the actual subtasks are attached. DB2 will deallocate these work data sets after the sort has completed.

DB2 utilities CHECK DATA, CHECK INDEX, CHECK LOB, LOAD, REBUILD INDEX, REORG TABLESPACE, and RUNSTATS have been enhanced to allocate sort work data sets dynamically before invoking DFSORT when:

- ▶ The new DSNZPARM UTSORTAL is set to YES.
- ▶ No SORTNUM value is specified in the utility statement.
- ▶ No DFSORT related DD cards (SORTWKnn, SW01WKnn, DATAWKnn, DA01WKnn, STATWKnn, ST01WKnn) are specified in the JCL.

Setting the new DSNZPARM IGNSORTN to YES will cause utilities to dynamically allocate sort work data sets even if the SORTNUM parameter was specified in the utility statement. The specified SORTNUM value will then be ignored. We recommend first changing a few representative utility jobs and remove the SORTNUM parameter to understand the possible changed execution behavior of those jobs. Once you are familiar and satisfied with this behavior, you can set IGNSORTN to YES as well to avoid having to change all existing jobs just to benefit from the new allocation logic.

The utility will try to allocate the required disk space in two data sets for each sort task, but will reduce the allocation size if the requested amount is not available. Allocation of additional data sets continues until the required amount is fully allocated (so it behaves like it starts with SORTNUM 2 and then increases the SORTNUM value dynamically until it works with a maximum of 255). Data sets will be allocated using DSNTYPE=LARGE if this setting supported by the system.

When the requested amount is not available, DFSMS failure messages such as “IGD17272I VOLUME SELECTION HAS FAILED FOR INSUFFICIENT SPACE FOR DATA SET...” will be issued before trying the smaller allocation. This is standard behavior and there is no reason for concern. DB2 could have suppressed these error messages, but provides these messages for your information. These message can be a trigger to review your sort pool volume size allocation.

Allocating sort work data sets will try to allocate data sets as large as possible and then reduce the allocation size if not successful. So if you have set up your sort pool in a way that uses large volumes with inferior performance in the overflow storage group, and only smaller volumes in your regular sort storage group, then these overflow volumes will be selected for most of the DB2 allocated sort work data sets, as they provide room for larger data sets than the regular volumes. You should review this sort pool setup when using the SORTNUM elimination feature by providing high performance large volumes.

For every invocation of DFSORT, the utility will pass an estimated number of records that need to be sorted. For some utilities, this number will be known exactly from previous phases, but most utilities have to estimate that number. Prior to this enhancement, that estimate was calculated using table space allocation information such as high used RBA and RUNSTATS values like AVGROWLEN and NPAGES. However, this technique depends on accurate statistics by periodically running the RUNSTATS utility.

By setting UTSORTAL=YES, the estimation will now first look up the processed objects in real-time statistics tables, which maintain current counts of rows or keys for every table space and every index. Only if the object cannot be found in RTS or the count is set to NULL will the previous estimation logic based on RUNSTATS output be used. Thus, the DB2 utilities enhancement enabled by UTSORTAL set to YES has a greater dependency on the accuracy of real-time statistics tables SYSIBM.SYSTABLESPACESTATS, and SYSIBM.SYSINDEXSPACESTATS.

However, we still recommend running RUNSTATS periodically:

- ▶ For those objects where the RTS are not available.
- ▶ REORG still uses the average row length from RUNSTATS to estimate the sort work data set size for the data sort.

The most likely reason why real-time statistics are not available or not usable is that the table space or index has not been reorganized or rebuilt since real-time statistics was enabled, as these operations will set the base line for RTS values. In some situations when RTS was already active in DB2 V8, there might be corrupted entries in the RTS tables that appear as very large. Values that are beyond table space or index space physical limits will be ignored. Those corrupted values could occur when there was an overflow into negative numbers before the PTF for PK53341 was installed on the system. To get valid RTS values, the affected table space or index has to be reorganized or rebuilt.

So keeping RTS accurate is beneficial, that is, activities that interfere with DB2 keeping the statistics up to date should be avoided, such as stopping the RTS table space or replacing objects outside of DB2. Also, RTS values may not have been externalized to the tables yet in the rare case of a DB2 crash, so that information may be lost. RTS are externalized with the STATSINT interval, so you might consider setting it to a smaller value instead of the default of 30 minutes in order to increase accuracy.

DB2 also uses a fall-back mechanism to derive values from other available real time statistics, for example, if the number of rows for a table space partition is needed, it will use the number of index entries for an associated index partition if it exists. This method cannot be used for XML indexes, as they can have a variable number of keys per row.

Multi-table table spaces can be a bit problematic, as there are no table specific real-time statistics and the overall table space statistics give no clue about how many rows are in each table. If available, then the index statistics for each contained table will be used to get a better idea of row distributions, but that requires that each contained table have at least one index with valid statistics. If no table statistics can be found, DB2 has to fall back to the old estimate where the total number of rows has to be divided by the number of tables.

When table spaces or indexes have been modified outside of DB2 (for example, using DSN1COPY or DFSMS copy), the real-time statistics maintained by DB2 will no longer be in synch with the current object status. To avoid misleading estimates for such objects, the related information should be invalidated to keep DB2 utilities from using it. For table spaces, set the column TOTALROWS in SYSIBM.TABLESPACESTATS to NULL, and for indexes, set the column TOTALENTRIES in SYSIBM.INDEXSPACESTATS to NULL. DB2 utilities will then use the estimates based on RUNSTATS information. For best results, real-time statistics should soon be re-initialized for such objects by running LOAD REPLACE, REORG, or REBUILD INDEX. Alternatively, you can set those columns to known values when you replace the table spaces using DSN1COPY. RTS will then accumulate all deltas to the newly set value.

When DB2 preallocates the work data sets, the DYNALOC option is no longer passed to DFSORT. The number of rows to be sorted is still passed to DFSORT by the FILSZ option. The values that are passed to DFSORT can be seen in message ICE000I in the corresponding DFSORT message data set, as shown in Example 6-16.

*Example 6-16 Determining the FILSZ parameter passed to DFSORT in the UTPRINnn message data set*

---

```

ICE200I 0 IDENTIFIER FROM CALLING PROGRAM IS 0001
ICE143I 0 BLOCKSET      SORT  TECHNIQUE SELECTED
ICE250I 0 VISIT http://www.ibm.com/storage/dfsor FOR DFSORT PAPERS, EXAMPLES AND MORE
ICE000I 0 - CONTROL STATEMENTS FOR 5694-A01, Z/OS DFSORT V1R10 - 15:21 ON TUE OCT 20, 2009 -
      SORT FIELDS=(00007.0,00020.0,A,00001.0,00005.0,A),FORMAT=BI,FILSZ=U0000*
      00001248795
      RECORD TYPE=F,LENGTH=(0026,0026,0026)
      OPTION MSGPRT=ALL,SORTDD=SW01,MSGDDN=UTPRIN01,MAINSIZE=MAX,USEWKDD
ICE201I F RECORD TYPE IS F - DATA STARTS IN POSITION 1
ICE751I 0 C5-K48846 C6-K90014 C7-K45047 C8-K46331 E4-BASE  C9-BASE  E5-K48846 E7-K48846
ICE193I 0 ICEAM2 INVOCATION ENVIRONMENT IN EFFECT - ICEAM2 ENVIRONMENT SELECTED
ICE089I 1 DB2R6Y .REORG      .      , INPUT LRECL = 26, TYPE = F
ICE093I 0 MAIN STORAGE = (MAX,33554432,33554432)
ICE156I 0 MAIN STORAGE ABOVE 16MB = (33497072,33497072)

```

---

It is still possible to override the FILSZ value passed to DFSORT by using a DFSPARM statement, as shown in Example 6-2 on page 146. Using the FILSZ OPTION statement (specified in MB), as described in APARs PK45916 and PK41899, should only be done when directed by IBM Support.

The usage of the new SORTNUM elimination feature can be verified by viewing the message DSNU3340I csect-name UTILITY PERFORMS DYNAMIC ALLOCATION OF SORT DISK SPACE (see Example 7-2 on page 172 and Example 8-2 on page 208).

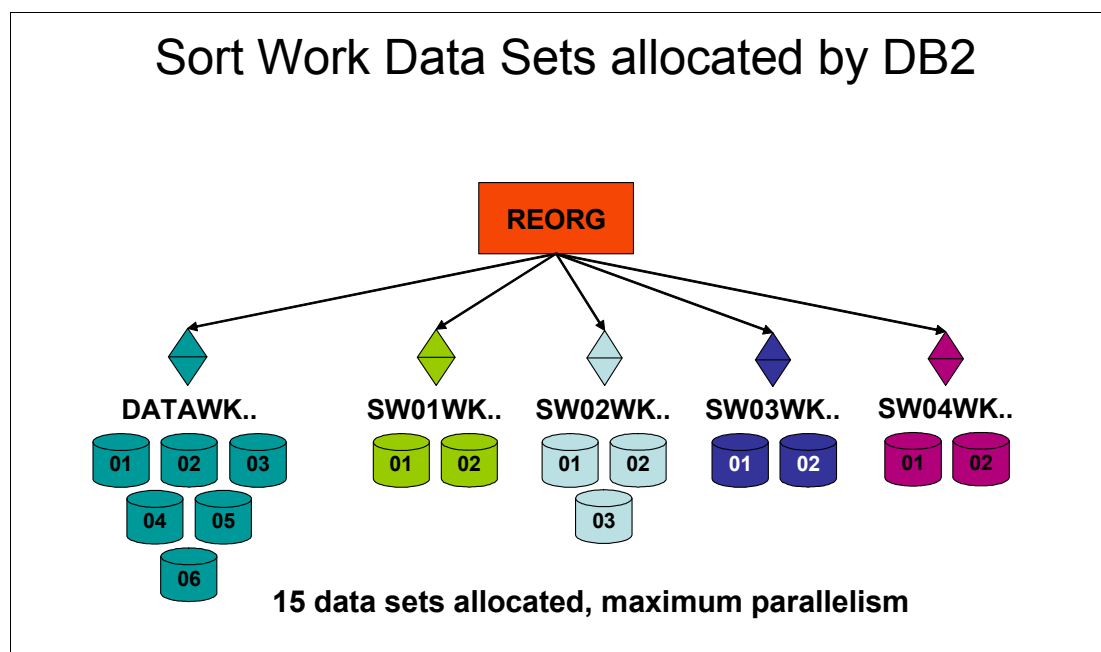
Finally, the utilities with parallel sorts have also been enhanced to allocate the sort work data sets for the largest sorts first, as long as the sort pool has the highest chance to satisfy those requests with large available chunks of disk space. The determination of the degree of parallelism is an iterative process where the number of data sets allocated so far is taken into account. Because the number of data sets allocated for a specific utility run may vary for every invocation, you may also see different degrees of parallelism for the same job, but DB2 will always attempt to use the available resources in the most efficient way.



# Sort Work Data Sets allocated by DFSORT

The diagram illustrates the allocation of 32 data sets by DFSORT. At the top, an orange box labeled "REORG" has four arrows pointing down to four categories: "DATAWK..", "SW01WK..", "SW02WK..", and "SW03WK..". Each category contains a pyramid of 8 cylinders labeled 01 through 08, colored teal, green, light blue, and dark blue respectively. The cylinders are arranged in a pyramid shape: 01, 02, 03 on the top row; 04, 05 on the second row; and 06, 07, 08 on the bottom row. Below the diagram, the text "32 data sets allocated, parallelism constrained" is displayed.

In Figure 6-3, we give the same example with the SORTNUM elimination feature. Here DB2 will allocate only 15 work data sets, and full parallelism with four parallel index sort tasks is possible.

Chapter 6. Sort processing **163**





## Part 3

# Executing DB2 utilities

In this part, we describe the functionality of the DB2 utilities and provide examples of their use in the following chapters:

- ▶ Chapter 7, “Loading and unloading data” on page 167
- ▶ Chapter 8, “Reorganizing data” on page 205
- ▶ Chapter 9, “Copying data” on page 223
- ▶ Chapter 10, “Recovering data” on page 251
- ▶ Chapter 11, “Gathering statistics” on page 291
- ▶ Chapter 12, “Verifying data consistency” on page 321
- ▶ Chapter 13, “BACKUP and RESTORE SYSTEM utilities” on page 339





## Loading and unloading data

The primary method for loading data is the LOAD utility. Its specific purpose is to populate DB2 tables with data. The input to LOAD can be a sequential data set, HFS data set, virtual file, or the result of an SQL query launched against tables on a local or remote server. With LOAD, you can get large amounts of data into DB2 tables as quickly as possible.

The online UNLOAD utility unloads data from one or more source objects to one or more sequential data set(s) or HFS file(s). The source can be DB2 table spaces or DB2 image copy data sets. The source cannot be a concurrent copy. UNLOAD must be run on the system where the definitions of the table space and the table exists. The output records that the UNLOAD utility writes are compatible as input to the LOAD utility; as a result, you can reload the original table or different tables.

With the LOAD utility, you can:

- ▶ Load rows into an empty table
- ▶ Add new rows to a table that is not empty
- ▶ Load rows into multiple tables within one table space
- ▶ Load into tables specifying a table alias or synonym (not view)
- ▶ Empty a table space before reloading the table(s)
- ▶ Load LOB and XML data
- ▶ Filter the input data using criteria supplied by the user
- ▶ Check the input data on unique, referential, and other constraint violations, and discard failing records to a sequential data set
- ▶ Load data using different concurrency levels

With the UNLOAD utility you can:

- ▶ Unload rows from one or more tables in the same table space
- ▶ Unload the base table or the clone table
- ▶ Unload rows from all tables from one or a list of table spaces
- ▶ Unload rows from a partition of a partitioned table space
- ▶ Unload rows from a table specifying a table alias or synonym (not view)
- ▶ Unload rows from a full image copy, incremental copy, or inline image copy
- ▶ Unload rows from a data set created with the stan-alone DSN1COPY utility
- ▶ Specify a subset of the columns and rows of the table to be unloaded and list the columns in a certain order
- ▶ Unload a sample of the data specified in percentage
- ▶ Limit the number of unloaded rows to a maximum
- ▶ Strip or truncate certain fields
- ▶ Convert fields to another data type and unload fields in external format
- ▶ Decide not to pad variable-length columns in the unloaded records to their maximum length
- ▶ Unload LOB and XML data
- ▶ Unload data using different concurrency levels
- ▶ Generate a punch file with LOAD control statements to be used by a consecutive LOAD utility

In this chapter, we mainly focus on new LOAD and UNLOAD options introduced in DB2 V8 and DB2 9:

- ▶ Best practices to get good LOAD and UNLOAD performance
- ▶ Loading into reordered row format
- ▶ Loading and unloading data in the delimited format
- ▶ Loading and unloading compressed data
- ▶ Using unicode control statements
- ▶ Loading and unloading unicode data
- ▶ Loading and unloading tables with multilevel security
- ▶ Load data with the NOCOPYPEND option
- ▶ Loading NOT LOGGED table spaces
- ▶ Unloading from a clone table
- ▶ Unloading from an image copy
- ▶ Loading and unloading decimal floating-point data
- ▶ Loading and unloading LOB and XML data
- ▶ Loading LOB data with SORTKEYS NO
- ▶ Cross loader considerations
- ▶ LOAD SHRLEVEL CHANGE and clone tables
- ▶ Skip locked data during unload
- ▶ Loading and unloading HFS files
- ▶ Loading and unloading virtual files (pipes)

In this chapter, we do not talk about other means of unloading data, such as the sample DSNTIAUL program, REORG UNLOAD EXTERNAL or the High Performance Unload (HPU) Tool.

HPU is not part of the DB2 Utilities Suite. It is a separately chargeable DB2 tool described in “DB2 High Performance Unload for z/OS” on page 455. It can also unload data from a table space or image copy. HPU can also work outside DB2, directly accessing the VSAM or sequential files that contain the table space or image copy data set. It can also unload selected rows and columns and can also generate load control statements. The syntax of HPU is more SQL oriented than the syntax of the UNLOAD utility. Compared to the UNLOAD utility elapsed time is normally comparable, whereas CPU consumption is typically less for HPU.

## 7.1 Best practices to obtain good LOAD and UNLOAD performance

Different options should be taken into consideration when trying to optimize the performance of the LOAD and UNLOAD utility.

### 7.1.1 Invoke parallelism

In Chapter 4, “Performance via parallelism of executions” on page 75, we explain how *parallelism* is the major technique that DB2 uses to shorten the elapsed times of utilities. Maximizing the exploitation of parallelism is of primary importance in reducing the elapsed time for the LOAD and UNLOAD utility.

Different forms of parallelism can be used with the LOAD and UNLOAD utility:

- ▶ Inline COPY and inline RUNSTATS in parallel subtasks, as discussed in 4.2.1, “Inline COPY” on page 77 and 4.2.2, “Inline RUNSTATS” on page 79
- ▶ Parallel Index Build (PIB) to rebuild the different indexes of the table space in parallel, as discussed in 4.4, “Parallelism with the LOAD utility” on page 82
- ▶ Parallel partition loading to load the partitions of a partitioned table space in parallel, as discussed in 4.4, “Parallelism with the LOAD utility” on page 82
- ▶ Parallel partition unloading to unload the partitions of a partitioned table space in parallel, as discussed in 4.5, “Partition parallelism with the UNLOAD utility” on page 84

If you want to minimize the elapsed times of your LOAD and UNLOAD jobs, try to use parallelism as much as possible:

- ▶ Use LOAD REPLACE with LOG NO and COPYDDN to minimize the log volume and to create an inline copy so that the table space is fully available to all applications afterwards and fully recoverable.
- ▶ Use LOAD REPLACE and LOAD RESUME NO with STATISTICS to keep the access path statistics current after LOAD or RELOAD.
- ▶ Provide a SORTKEYS value when the input data set is on tape or a PDS member or a HFS file or a virtual file or when using a cursor to assure Parallel Index Build.
- ▶ Use one input data set per partition if the table is partitioned to enable parallel partition loading.
- ▶ Use one output data set per partition if the table is partitioned to enable parallel partition unloading.
- ▶ Let DB2 allocate the optimal number of sort data sets during the index build by eliminating the SORTNUM parameter and by removing sort workfiles from the JCL. Refer to Chapter 6, “Sort processing” on page 141 for more details and other considerations concerning SORT.
- ▶ Check if the degree of parallelism is not constrained by monitoring the DSNU3971 message in the job output of the LOAD utility, as explained in 4.9, “Considerations for running parallel subtasks” on page 90.



In Example 7-1 we show a LOAD REPLACE of a segmented table space using Parallel Index Build (PIB). The PIB is invoked by default. SORTKEYS is not specified and estimated upon the number of records in the input sequential data set. We use inline copy and inline RUNSTATS, as recommended. We do not specify SORTNUM and let DB2 allocate the optimal number of sort data sets. We have no workfiles allocated in the JCL.

*Example 7-1 LOAD of a segmented table space with Parallel Index Build*

---

```

TEMPLATE TSYSREC DSN('&US..MTSTBX.ORIG.DATA')
      DISP(OLD,KEEP,KEEP)
TEMPLATE TSORTOUT DSN('&US..&SS..&DB..&SN..S&JU(3,5)..#&TI.')
      SPACE TRK MAXPRIME 65536
      DISP(MOD,DELETE,CATLG)
TEMPLATE TSYSUT1 DSN('&US..&SS..&DB..&SN..U&JU(3,5)..#&TI.')
      DISP(MOD,DELETE,CATLG)
TEMPLATE TSYSCOPY
      DSN('&US..&SS..&DB..&SN..&IC.&JU(3,5)..#&TI.')
      SPACE TRK MAXPRIME 65536
      DISP(MOD,CATLG,CATLG)
LOAD DATA INDDN TSYSREC LOG NO REPLACE
      EBCDIC CCSID(01148,00000,00000)
      WORKDDN(TSYSUT1,TSORTOUT) SORTDEVT 3390
      STATISTICS TABLE(ALL) INDEX(ALL KEYCARD)
      INTO TABLE "S"."MTSTBP"
      WHEN(00001:00002) = X'0003'
      ( "COD_DL_MAT"
        POSITION( 00003:00010) CHAR(00008)
      , "NR_DL_MAT"
        POSITION( 00011:00036) TIMESTAMP EXTERNAL(026)
      , "TY_DL_MAT"
        POSITION( 00037:00037) CHAR(00001)
      .....
      .....
      , "TE_LN_STRC"
        POSITION( 00228:00231) INTEGER
      , "TE_RH_STRC"
        POSITION( 00232:00235) INTEGER
      )
      /*

```

---

An extract of the job output can be seen in Example 7-2. We have no DSNU397I message and the maximum degree of parallelism was used.

*Example 7-2 Job output of LOAD with Parallel Index Build*

---

```

DSNU3340I   280 17:15:24.45 DSNURPIB - UTILITY PERFORMS DYNAMIC ALLOCATION OF SORT DISK SPACE
DSNU3342I   280 17:15:24.91 DSNURPIB - NUMBER OF OPTIMAL SORT TASKS = 5, NUMBER OF ACTIVE SORT TASKS = 5
DSNU395I    280 17:15:24.91 DSNURPIB - INDEXES WILL BE BUILT IN PARALLEL, NUMBER OF TASKS = 15
DSNU304I   -DB9A 280 17:15:52.12 DSNURWT - (RE)LOAD PHASE STATISTICS - NUMBER OF RECORDS=1248796 FOR TABLE S.MTSTBP
DSNU1147I   -DB9A 280 17:15:52.12 DSNURWT - (RE)LOAD PHASE STATISTICS - TOTAL NUMBER OF RECORDS LOADED=1248796 FOR
TABLESPACE MTSTBP.MTSTBP
DSNU302I    280 17:15:52.13 DSNURILD - (RE)LOAD PHASE STATISTICS - NUMBER OF INPUT RECORDS PROCESSED=1248796
DSNU300I    280 17:15:52.13 DSNURILD - (RE)LOAD PHASE COMPLETE, ELAPSED TIME=00:00:28
DSNU394I   -DB9A 280 17:15:56.44 DSNURBXA - SORTBLD PHASE STATISTICS - NUMBER OF KEYS=1248796 FOR INDEX S.I_MTSTBP_5
DSNU394I   -DB9A 280 17:15:56.84 DSNURBXA - SORTBLD PHASE STATISTICS - NUMBER OF KEYS=1248796 FOR INDEX S.I_MTSTBP_4
DSNU394I   -DB9A 280 17:15:56.84 DSNURBXA - SORTBLD PHASE STATISTICS - NUMBER OF KEYS=1248796 FOR INDEX S.I_MTSTBP_3
DSNU394I   -DB9A 280 17:15:57.97 DSNURBXA - SORTBLD PHASE STATISTICS - NUMBER OF KEYS=1248796 FOR INDEX S.I_MTSTBP_1
DSNU394I   -DB9A 280 17:15:58.06 DSNURBXA - SORTBLD PHASE STATISTICS - NUMBER OF KEYS=1248796 FOR INDEX S.I_MTSTBP_2
DSNU391I    280 17:15:58.21 DSNURPTB - SORTBLD PHASE STATISTICS. NUMBER OF INDEXES = 5
DSNU392I    280 17:15:58.21 DSNURPTB - SORTBLD PHASE COMPLETE, ELAPSED TIME = 00:00:06

```

---

Example 7-3 shows a LOAD REPLACE of a partitioned table space using parallel partition loading and parallel index build (PIB). The parallel partition loading is invoked by having one input data set per partition, which is specified in the TSYSREC template using the &PART. or &PA. object variable and one INTO TABLE PART statement per partition. The PIB is invoked by default. We also use inline copy and inline RUNSTATS, as recommended. SORTKEYS is not specified and estimated upon the number of records in the input sequential data sets. We do not specify SORTNUM and let DB2 allocate the optimal number of sort data sets. We have no workfiles allocated in the JCL.

*Example 7-3 Load of a partitioned table space with full parallelism*

---

```

TEMPLATE TSYSREC DSN('DB2R6.DB9A.MTSTB0.MTSTB0.R09280.#&PA.')
      DISP(OLD,KEEP,KEEP)
TEMPLATE TSORTOUT DSN('&US..&SS..&DB..&SN..S&JU(3,5)..#&TI.')
      SPACE TRK MAXPRIME 65536
      DISP(MOD,DELETE,CATLG)
TEMPLATE TSYSUT1 DSN('&US..&SS..&DB..&SN..U&JU(3,5)..#&TI.')
      DISP(MOD,DELETE,CATLG)
TEMPLATE TSYSCOPY
      DSN('&US..&SS..&DB..&SN..&IC.&JU(3,5)..#&TI.')
      SPACE TRK MAXPRIME 65536
      DISP(MOD,CATLG,CATLG)
LOAD DATA LOG NO REPLACE
      EBCDIC CCSID(01148,00000,00000)
      COPYDDN(TSYSCOPY) WORKDDN(TSYSUT1,TSORTOUT) SORTDEVT 3390
      STATISTICS TABLE(ALL) INDEX(ALL KEYCARD)
      INTO TABLE "S"."MTSTB0"
      PART 00001 INDDN TSYSREC
      WHEN(00001:00002) = X'0003'
      ( "COD_DL_MAT"
        POSITION( 00003:00010) CHAR(00008)
      , "NR_DL_MAT"
      .....
      .....
      , "TE_RH_STRC"
        POSITION( 00232:00235) INTEGER
      )
      INTO TABLE "S"."MTSTB0"

```

```

PART 00002 INDDN TSYSREC
WHEN(00001:00002) = X'0003'
( "COD_DL_MAT"
  POSITION( 00003:00010) CHAR(00008)
, "NR_DL_MAT"
.....
.....
, "TE_RH_STRC"
  POSITION( 00232:00235) INTEGER
)
INTO TABLE "S"."MTSTB0"
PART 00025 INDDN TSYSREC
WHEN(00001:00002) = X'0003'
( "COD_DL_MAT"
  POSITION( 00003:00010) CHAR(00008)
, "NR_DL_MAT"
.....
.....
, "TE_RH_STRC"
  POSITION( 00232:00235) INTEGER
)
/*

```

---

An extract of the job output can be seen in Example 7-4.

*Example 7-4 Job output of LOAD with full parallelism*

---

```

DSNU350I -DB9A 280 15:13:37.89 DSNURRST - EXISTING RECORDS DELETED FROM TABLESPACE
DSNU3340I 280 15:13:37.92 DSNURPIB - UTILITY PERFORMS DYNAMIC ALLOCATION OF SORT DISK SPACE
DSNU3342I 280 15:13:38.22 DSNURPIB - NUMBER OF OPTIMAL SORT TASKS = 5, NUMBER OF ACTIVE SORT TASKS = 3
DSNU395I 280 15:13:38.22 DSNURPIB - INDEXES WILL BE BUILT IN PARALLEL, NUMBER OF TASKS = 9
DSNU364I 280 15:13:38.23 DSNURPPL - PARTITIONS WILL BE LOADED IN PARALLEL, NUMBER OF TASKS = 3
DSNU397I 280 15:13:38.23 DSNURPPL - NUMBER OF TASKS CONSTRAINED BY CPUS
DSNU1148I 280 15:13:40.42 DSNURPLD - (RE)LOAD PHASE STATISTICS - NUMBER OF INPUT RECORDS PROCESSED=224 FOR PART 2
DSNU1148I 280 15:13:40.42 DSNURPLD - (RE)LOAD PHASE STATISTICS - NUMBER OF INPUT RECORDS PROCESSED=2982 FOR PART 3
DSNU1148I 280 15:13:42.80 DSNURPLD - (RE)LOAD PHASE STATISTICS - NUMBER OF INPUT RECORDS PROCESSED=18 FOR PART 4
DSNU1148I 280 15:13:42.80 DSNURPLD - (RE)LOAD PHASE STATISTICS - NUMBER OF INPUT RECORDS PROCESSED=160767 FOR PART 1
DSNU1148I 280 15:13:45.17 DSNURPLD - (RE)LOAD PHASE STATISTICS - NUMBER OF INPUT RECORDS PROCESSED=10335 FOR PART 7
.....
DSNU610I -DB9A 280 15:14:14.33 DSNUSUCD - SYSCOLDIST CATALOG UPDATE FOR S.I_MTSTB0_3 SUCCESSFUL
DSNU610I -DB9A 280 15:14:14.33 DSNUSUCD - SYSCOLDIST CATALOG UPDATE FOR S.I_MTSTB0_5 SUCCESSFUL
DSNU620I -DB9A 280 15:14:14.33 DSNUSEOF - RUNSTATS CATALOG TIMESTAMP = 2009-10-07-15.13.37.897469
DSNU428I 280 15:14:14.71 DSNURELD - DB2 IMAGE COPY SUCCESSFUL FOR TABLESPACE MTSTB0.MTSTB0
DSNU010I 280 15:14:14.86 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0

```

---

In message DSNU397I, we see that parallelism is constrained by the number of available CPUs on the LPAR. The utility could start additional tasks, but it would not be efficient to do so, because it would overtax the available CPUs.

The easiest way to create the input utility statement for LOAD is by using the punch file generated by the UNLOAD utility on the same table space. You can use an UNLOAD statement, as shown in Example 7-5, using the PARTLEVEL keyword for unloading the different partitions in parallel to different output data sets. Use the &PART. keyword to differentiate the output data sets. You will probably need to change the generated punch file before using it with the LOAD (adding templates, adding the STATISTICS keyword, and adjusting some other options).

*Example 7-5 UNLOAD statement to generate the input statements for LOAD*

---

```

TEMPLATE TSYPUN DSN('&US.&SS.&DB.&SN..P&JU(3,5)..PUNCH')
      DISP(MOD,CATLG,CATLG)
TEMPLATE TSYSREC DSN('&US.&SS.&DB.&SN..R&JU(3,5)..#&PART. ')
      DISP(MOD,CATLG,CATLG)
LISTDEF UNLIST INCLUDE TABLESPACE MTSTBO.MTSTBO PARTLEVEL
UNLOAD LIST UNLIST
UNLDDN(TSYSREC) PUNCHDDN(TSYPUN)
/*

```

---

**Note:** Prior to Version 8, a page set REBUILD PENDING status on a non-partitioning index blocked the ability to LOAD REPLACE any partition of the table. This is no longer the case, but the NPI will remain in the REBUILD PENDING state.

## 7.1.2 Use the REUSE option with LOAD

You can use the REUSE option with LOAD to logically reset and reuse DB2-managed data sets without deleting and redefining them. It can only be used with the LOAD REPLACE option and with STOGROUP defined data sets. Normally, if you do not specify REUSE, LOAD REPLACE of a DB2-managed data set will delete and redefine the underlying VSAM data sets. With the REUSE option, the underlying VSAM data sets will not be deleted and redefined, but only logically reset; that means the high used RBA is reset back to zero.

The logical reset time of a VSAM data set is much shorter than the physical delete and redefine time. Bear in mind that the logical reset of a VSAM data set will not reclaim any secondary extents.

The REUSE option can operate on the entire table space and its index spaces, or on a partition of a partitioned table space and its corresponding partition index space. In the last case, you should specify PART integer REPLACE REUSE:

- ▶ To logically reset and reuse an entire table space and associated index spaces, use:  
LOAD DATA REPLACE REUSE INTO TABLE table name
- ▶ To logically reset and reuse a table space partition and associated index partition, use:  
LOAD DATA INTO TABLE table name PART 1 REPLACE REUSE

We recommend using the REUSE option:

- ▶ If you want to preserve the allocated space on the disk volumes between consecutive LOAD REPLACE operations (for I/O tuning reasons or to avoid space problems in case of reallocation).
- ▶ To reduce CPU and elapsed times, if you have partitioned table spaces with a lot of underlying VSAM data sets.

Do *not* use the REUSE option if you want to do any of the following tasks:

- ▶ Reduce the number of secondary extents.
- ▶ Activate altered PRIQTY and SECQTY values.
- ▶ Move the data sets to another STOGROUP.

### 7.1.3 Other things that will influence the performance

Other actions that will influence the performance of the LOAD utility are:

- ▶ Use one LOAD DATA statement when loading multiple tables in the same table space. Follow the LOAD statement with multiple INTO TABLE WHEN clauses.
- ▶ Avoid data conversions, such as from integer to decimal or from decimal to floating-point. Also, load numeric data in its internal representation when available.
- ▶ Sort the input data in cluster order prior to load to avoid needing to reorganize it after loading.
- ▶ Try to avoid having the input data set on tape when using the DISCARD option, because input is read twice to discard the errant records.
- ▶ Enable Parallel Access Volume (PAV) on your DASD devices.
- ▶ If you choose to compress your data, specify KEEPDICTIONARY to avoid having DB2 rebuilding the compression dictionary.
- ▶ Be sure to have the following maintenance applied: PK47083, PK60956, PK61759, PK70866, PK75216 (DB2 9 only), PK76860, PK79136, and PK85889.

Actions that will influence the performance of the UNLOAD utility are:

- ▶ Avoid data conversions. Unload all the columns with the same data types as the columns of the source table.
- ▶ Do not specify a column list.
- ▶ SHRLEVEL REFERENCE will be faster than SHRLEVEL CHANGE (APAR PK60956).
- ▶ Specify ISOLATION UR if SHRLEVEL CHANGE.
- ▶ Use NOPAD for variable length column output.
- ▶ If there are many tables in the table space, be sure to have PK77313 applied. This will use the DBD instead of the catalog for retrieving the column information of the tables.

## 7.2 Loading into reordered row format

Initially, when in DB2 9 NFM, data that was loaded with the LOAD REPLACE utility was automatically loaded in the new *reordered row format (RRF)*. The same was true when using REORG on an existing table space. Newly created table spaces were always created in RRF when in DB2 9 NFM.

This new format for data rows was introduced in DB2 9 to reduce the impact of processing variable length columns. Prior to DB2 9 for z/OS, each variable length column in a data row was stored with its length preceding it. This format is called *basic row format (BRF)*. If you needed to access a column that was preceded by a variable length column, DB2 had to traverse through all the columns that preceded it, starting with the first variable length column until it reached the column you wanted to process. If you updated a variable length column, DB2 logged the changed column and everything to the end of the row. For these reasons, we used to recommend that all variable length columns be placed at the end of the row and the most frequently changed ones at the very end of the row.

With RRF, you do not have to worry about the order of columns within the row. You can specify the order however you wish, and DB2 automatically reorders the columns within the row and places all of the variable length columns at the end of the physical row within the data page. Instead of storing the columns with their length, DB2 stores all of the offsets for the variable length columns in an area following the last fixed length column and prior to the first variable length column. DB2 can now directly access each variable length column and know its length by doing simple calculations with the offsets. In most cases, this is a much more efficient way to process individual varying length columns.

Initially, when you ran LOAD REPLACE on a table space or partition that was in basic row format, LOAD REPLACE converted the table space or partition to the reordered row format when in DB2 9 NFM. If the clause EDITPROC or VALIDPROC was used in a table space or partition, the table space or partition remained in basic format after the LOAD REPLACE.

The same was true for compressed table spaces. LOAD REPLACE and REORG changed the row format from BRF to RRF and, by default, a new dictionary was always built, regardless of the KEEPDICTIONARY option. With subsystem parameter HONOR\_KEEPPDICTIONARY = YES, it was possible to honor the KEEPDICTIONARY option when converting to RRF. This behavior changed with PK78958. From now on, compressed table spaces in BRF were never converted from BRF to RRF because some cases were found where the compression factor was worse in RRF than in BRF (especially when there are a lot of very small VARCHARs in the data).

APAR PK87348 introduces a more flexible way of dealing with BRF and RRF. It introduces a new subsystem parameter SPRMRRF and a new LOAD UTILITY option ROWFORMAT that lets you control the use of BRF and RRF.

LOAD REPLACE and REORG TABLESPACE will honor the SPRMRRF setting in DB2 9 NFM as follows:

- ▶ When SPRMRRF=ENABLE (the default value), eligible table space will be converted from BRF to RRF. Newly created table spaces, or newly added partitions on Partitioned By Growth (PBG) table space and by ALTER ADD PARTITION, will be created in RRF. For an existing BRF table space containing table(s) with EDITPROC, a newly created partition will remain in BRF.

- ▶ When `SPRMRRF=DISABLE`, eligible table space will not be converted from BRF to RRF, that is, if the table space is in BRF prior to `LOAD` or `REORG`, it will remain in BRF after the utility completes; similarly, if the table space is in RRF, it will stay in RRF. Newly created table spaces, including Universal Table Spaces (UTS), and newly added partitions on PBG and by `ALTER ADD PARTITION`, will be created in BRF. XML table spaces will always be in RRF
- ▶ The compression attribute of the table space is no longer considered for row format conversion, that is, a `COMPRESS YES` or `COMPRESS NO` BRF table space can be converted to RRF in DB2 9 NFM by `LOAD` or `REORG` when `SPRMRRF=ENABLE`.

`LOAD REPLACE` and `REORG` will honor the new `ROWFORMAT` option in DB2 9 NFM as follows:

- ▶ This keyword overrides the existing `SPRMRRF` setting when specified, but has no effect on LOB, catalog, directory, XML, and universal table spaces participating in a `CLONE` relationship.
- ▶ `ROWFORMAT BRF` specifies that the table space or partition(s) being reorganized or replaced will be converted to or left in Basic Row Format (BRF).
- ▶ `ROWFORMAT RRF` specifies that the table space or partition(s) being reorganized or replaced will be converted to or left in Reordered Row Format (RRF).

When the row format is changed by `LOAD` or `REORG`, message `DSNU1173I` will be written, as shown in Example 7-6. Furthermore, the column `TTYPE` of `SYSIBM.SYSCOPY` will contain the values 'RRF' or 'BRF' for the row that corresponds with the utility.

#### *Example 7-6 Converting the row format*

---

```
DSNU1173I -DB9A 289 12:48:52.90 DSNURFIT - TABLE SPACE MTSTBO.MTSTBO WILL BE CONVERTING FROM BASIC ROW
FORMAT TO REORDERED ROW FORMAT
```

```
DSNU1173I -DB9A 289 12:57:15.37 DSNURWI - TABLE SPACE MTSTBP.MTSTBP WILL BE CONVERTING FROM REORDERED ROW
FORMAT TO BASIC ROW FORMAT
```

---

There are different reasons to set `SPRMRRF=DISABLE`:

- ▶ During migration time of your different subsystems to DB2 9 NFM. If you use `DSN1COPY` to copy data between different subsystems, you should wait till all subsystems are in NFM before setting `SPRMRRF=ENABLE`. Ensure that source and target are both in BRF or both in RRF when executing the `DSN1COPY`.
- ▶ When the compression factor of compressed table spaces is worse in RRF than in BRF. In that case, you can set `SPRMRRF=DISABLE` and migrate your data back to BRF using `REORG` with `ROWFORMAT BRF`.

We recommend only using `SPRMRRF=DISABLE` or the `ROWFORMAT BRF` keyword during transition time to DB2 9 NFM or when you encounter a significant compression ratio reduction with the RRF format. In all other cases, use `SPRMRRF=ENABLE`.

The actual format of a table space or partition can be seen in column `FORMAT` of `SYSIBM.SYSTABLEPART` ('R' for RRF, ' ' for BRF). Also, the column `TTYPE` of `SYSIBM.SYSCOPY` will contain the values BRF' or 'RRF' after a `LOAD` or `REORG` that changed the row format.

**Attention:** Currently, when using a DSN1COPY of a table space to populate another table space, the user needs to be sure that the row formats of the DSN1COPY and the table space to be populated match. The results of using a DSN1COPY of a table space in Reordered Format to populate a table space that is in basic row format or of using a table space in basic row format to populate a table space in Reordered Format could cause the utilities to work incorrectly because of the mismatch between control information and format of the data. This is true for any other type of mismatch.

**Tip:** Help is on the way with APAR PM03691, which is OPEN at the time of writing.

## 7.3 Loading and unloading data in delimited format

Since DB2 V8, the LOAD utility is enhanced to accept data from a delimited file specifying the LOAD DELIMITED option. The UNLOAD utility is also enhanced to produce a delimited file when unloading the data. These enhancements help to simplify the process of moving data into and out of DB2 for z/OS. Before this time, you were obliged to write a program to convert the delimited data into the positional format that the LOAD utility understood or to convert the output data of UNLOAD to delimited format.

Most other relational databases like DB2 on Linux®, UNIX, and Windows® and Oracle can unload data in delimited format, where each record is a row, and columns are separated by commas, for example. So this enhancement eased dramatically the import or export of a large amount of data from other operating system platforms to DB2 for z/OS and vice versa. Also, it became much easier to move data from a spreadsheet on a workstation to DB2 and vice versa.

For example, if you want to move data from a spreadsheet on a Windows workstation to DB2 for z/OS, you need to take the following steps:

1. Create the target DB2 table as an EBCDIC or an ASCII table.
2. Save the spreadsheet data in comma separated value (CSV) format.
3. Upload the saved data to the host in one of the following formats using FTP or the terminal emulator's file transfer program:
  - Text (The data is converted from ASCII to EBCDIC before it is stored on the host.)
  - Binary (The data is uploaded in ASCII.)
4. Use the LOAD utility with the FORMAT DELIMITED EBCDIC (text) or FORMAT DELIMITED ASCII (binary) option.

If you want to move data from DB2 to a spreadsheet or other database system:

1. Use the UNLOAD utility with FORMAT DELIMITED EBCDIC or ASCII.
2. Transmit the data to the workstation in text or binary format.
3. Load/import the file in the spreadsheet or database.



When data is in a delimited format, all fields in the input data set are character strings or external numeric values. In addition, each column in a delimited file is separated from the next column by a column delimiter character, as shown in Example 7-7.

*Example 7-7 Delimited format*

---

```
350,"GAFNEY",84,"CLERK",5,13030.50,188.00
340,"EDWARDS",84,"SALES",7,17844.00,1285.00
330,"BURKE",66,"CLERK",1,10988.00,55.50
320,"GONZALES",66,"SALES",4,16858.20,844.00
310,"GRAHAM",66,"SALES",13,21000.00,200.30
300,"DAVIS",84,"SALES",5,15454.50,806.10
290,"QUILL",84,"MGR",10,19818.00,
280,"WILSON",66,"SALES",9,18674.50,811.50
270,"LEA",66,"MGR",9,18555.50,
260,"JONES",10,"MGR",12,21234.00,
```

---

Because of the conversion of the external numeric values and the scan of the varchar fields to determine the length of the field, loading delimited data will be more expensive than loading data from a positional format, so use the `FORMAT DELIMITED` option only when needed.

For the delimited data, you can specify the column delimiter `COLDEL` (the default is the comma), the string delimiter `CHARDEL` (the default is the double quotation mark), and the decimal point character `DECPT` (the default is the period). The delimiters can be specified as characters or hexadecimal values and must match the codepage of the input data.

For delimited output, `UNLOAD` does not add trailing padded blanks to variable-length columns, even if you do not specify the `NOPAD` option. For fixed-length columns, the normal padding rules apply.

We recommend using Unicode when the input or output delimited file is coming or going from or to a system other than z/OS or from a DB2 for z/OS system that has a different EBCDIC or ASCII CCSID. We do this task to avoid possible CCSID translation problems. The default delimiters in ASCII or Unicode are `X'2C'` for `COLDEL`, `X'22'` for `CHARDEL`, and `X'2E'` for the `DECPT`. Unicode input data for `FORMAT DELIMITED` must be UTF-8, CCSID 1208.

For more details about loading and unloading delimited files, refer to *DB2 Version 9.1 for z/OS Utility Guide and Reference*, SC18-9855.

## 7.4 Loading and unloading compressed data

When loading data into a table space defined with the `COMPRESS YES` keyword, a compression dictionary is built. `RESUME YES` will only build a dictionary if the table space is empty. After the dictionary is completely built, the rest of the data is compressed as it is loaded. For normal partitioned table spaces and partition-by-range table spaces, a compression dictionary will be built and stored in each partition. For partition-by-growth table spaces, the utility only builds one dictionary and the same dictionary page is populated through all partitions. For XML table spaces that are defined with `COMPRESS YES`, compression does not occur until the first `REORG`.

The data is not compressed until the dictionary is built. You must use `LOAD REPLACE` or `RESUME NO` to build the dictionary, except for simple table spaces where `LOAD REPLACE` must be used to build new compression dictionaries. To save processing costs, an initial `LOAD` does not go back to compress the records that were used to build the dictionary.

The number of records that are required to build a dictionary is dependent on the frequency of patterns in the data. For large data sets, the number of rows that are required to build the dictionary is a small percentage of the total number of rows that are to be compressed. For the best compression results, build a new dictionary whenever you first REORG the data.

When loading data into a compressed table space, two options can be specified regarding the compress dictionary:

- ▶ KEEPDICTIONARY
- ▶ COPYDICTIONARY

We recommend using the *sliding scale algorithm* when allocating compressed table spaces. Let DB2 take care of the space allocations.

There is also a limit to the maximum number of compressed partitions you can LOAD in a single LOAD statement if you do not use parallel partition loading by specifying the PART keyword in the INTO TABLE statement for each partition. This limit is controlled by the subsystem parameter MAX\_UTIL\_PARTS, which has a default of 254 and can have a maximum of 4096 (introduced by APAR PK51853).

When you unload data from a compressed table space, the compressed data rows will be automatically uncompressed using the compression dictionary. The same process occurs when unloading from a full image copy, inline copy, or incremental copy taken with the SYSTEMPAGES option. If the specified image copy data set is an incremental image copy or a copy of a partition or partitions, you can unload compressed records only when the same data set contains the dictionary pages for decompression. If an image copy data set contains a compressed row and a dictionary is not available, DB2 issues an error message.

**Attention:** The unloaded data from a compressed table space can be much bigger than the compressed data. Be sure to allocate an unload data set that is big enough to hold the unloaded data. Eventually, use DFSMS compress to compress the sequential data set or unload to a tape data set.

### 7.4.1 KEEPDICTIONARY

This option prevents the LOAD utility from building a new compression dictionary. LOAD retains the current compression dictionary and uses it for compressing the input data. This option eliminates the cost that is associated with building a new dictionary. You can use it when you are satisfied with the compression you are getting from the existing dictionary.

If the table space or partition is empty, DB2 performs one of these actions:

- ▶ DB2 builds a dictionary if a compression dictionary does not exist.
- ▶ DB2 keeps the dictionary if a compression dictionary exists.

If RESUME NO and REPLACE are specified when the table space or partition is not empty, DB2 performs the same actions as it does when the table space or partition is empty.

If the table space or partition is not empty and RESUME YES is specified, DB2 performs one of these actions:

- ▶ DB2 does not build a dictionary if a compression dictionary does not exist.
- ▶ DB2 keeps the dictionary if a compression dictionary exists.

The KEEPDICTIONARY keyword is ignored for XML table spaces. If you specify REPLACE, any existing dictionary for the XML table space or partition is deleted. If you do not specify REPLACE, any existing dictionary for the XML table space or partition is saved. DB2 does not compress an XML table space during the LOAD process. If the XML table space is defined with COMPRESS YES, the XML table space is compressed during REORG.

Consider using KEEPDICTIONARY if the last dictionary was built by REORG; the REORG utility's sampling method can yield more representative dictionaries than LOAD and can thus mean better compression.

## 7.4.2 COPYDICTIONARY

This new keyword in DB2 9 allows compression dictionaries to be copied from one partition to another in a classic partitioned or partition-by-range table space. This option provides a method to copy a compression dictionary to an empty partition that normally would not have a compression dictionary built yet. This option can only be used with the RESUME NO option at the table space level with PART integer REPLACE statements, where the partition where the dictionary is copied from is not the same as the partition(s) being replaced. Example 7-8 shows how to copy a compression dictionary from physical partition 1 to partitions 3 and 5.

*Example 7-8 Use of COPYDICTIONARY to copy a compression dictionary*

---

```
LOAD RESUME NO COPYDICTIONARY 1
INTO TABLE PART 3 REPLACE
INTO TABLE PART 5 REPLACE
```

---

The COPYDICTIONARY keyword is incompatible with the KEEPDICTIONARY keyword at the table space or partition level and the REPLACE keyword at the table space level.

## 7.4.3 Using SLIDING SCALE

One of the things that is difficult for a user when using compressed table spaces is estimating a good size for the PRIQTY and SECQTY quantities before the LOAD. When using the sliding scale algorithm, this is no longer necessary. Just specify PRIQTY=-1 and SECQTY=-1 or omit the PRIQTY and SECQTY values during the CREATE TABLESPACE and CREATE INDEX statements and make sure that the subsystem parameter OPTIMIZE EXTENT SIZING or MGEXTSZ on installation panel DSNTIP7 is set to YES.

When MSGEXTSZ=YES, DB2 will allocate the DB2 extents using a sliding scale algorithm. The primary extent will be allocated with a default value (1 CYL for non-LOB table spaces and indexes, 10 CYLS for LOB table spaces). The secondary extents will be allocated with an increased size. This function is also called *autonomic DB2 space allocation*. The algorithm helps to reduce the number of wasted space and improves the user productivity.

For more information about space allocations and the sliding scale algorithm, refer to *DB2 Version 9.1 for z/OS Administration Guide*, SC18-9840 and the IBM Redpapers™ publication *IBM System p5 520 and 520Q Technical Overview and Introduction*, REDP-4137.

## 7.5 Loading and unloading Unicode data

You can use input files in Unicode to LOAD a table space. Use the UNICODE option to specify that the input file is in UNICODE. You can also specify three CCSID values that correspond to the Unicode CCSID for SBCS, the Unicode CCSID for mixed DBCS data, and the Unicode CCSID for DBCS data. If you do not specify CCSID values, the default CCSID values are the Unicode CCSIDs that are specified at system installation. If the CCSIDs of the input data file do not match the CCSIDs of the table that is being loaded, the input data is converted to the table CCSIDs before being loaded.

Input fields with data types CHAR, CHAR MIXED, CLOB, DBCLOB, VARCHAR, VARCHAR MIXED, GRAPHIC, GRAPHIC EXTERNAL, and VARGRAPHIC are converted from the CCSIDs of the input file to the CCSIDs of the table space when they do not match. For example:

- ▶ You specify the UNICODE option for the input data, and the table space is EBCDIC.
- ▶ You specify the UNICODE option and the table space is ASCII.
- ▶ You specify the UNICODE option with the CCSID option and the CCSIDs are not the same as the CCSIDs of the table space.

An example of the LOAD statement with Unicode input data set is shown in Example 7-9.

*Example 7-9 Loading Unicode data*

---

```
LOAD DATA INDDN REC1 LOG YES REPLACE
UNICODE CCSID(00367,01208,01200)
INTO TABLE "ADMFO01 " ."TBMG0301"
WHEN(00004:00005 = X'0003')
```

---

You must code UNICODE constants in the WHEN, NULLIF and DEFAULTIF clauses.

You can also UNLOAD data into Unicode format by specifying the UNICODE option. If the source data is in a different encoding scheme like EBCDIC, the data is then converted into Unicode. You can also specify the three CCSID values that are to be used for the data of character type in the output records, including data that is unloaded in the external character format. These are the Unicode CCSID for SBCS, the Unicode CCSID for mixed DBCS data, and the Unicode CCSID for DBCS data. If you do not specify CCSID values, the default CCSID values are the Unicode CCSIDs that are specified at system installation.

When a CCSID conversion is requested, CCSID character substitutions can occur in the output string. Use the NOSUBS option to prevent possible character substitutions during CCSID conversion. If you specify the NOSUBS keyword and character substitution is attempted while data is being unloaded, this action is treated as a conversion error. The record with the error is not unloaded, and the process continues until the total error count reaches the number that is specified by the MAXERR option.

## 7.6 Loading and unloading tables with multilevel security

Multilevel security allows you to classify objects and users with security labels that are based on hierarchical security levels and non-hierarchical security categories. Multilevel security prevents unauthorized users from accessing information at a higher classification than their authorization, and it prevents users from declassifying information. Using multilevel security with row-level granularity, you can define security for DB2 objects and perform security checks, including row-level security checks. Row-level security checks allow you to control which users have authorization to view, modify, or perform other actions on specific rows of data.

If you use RACF® access control with multilevel security, you need additional authorizations to run LOAD jobs on tables that have multilevel security with row-level granularity. You must have an accessible valid security label and meet the following authorization requirements:

- ▶ To replace an entire table space with LOAD REPLACE, you must have the write-down privilege unless write-down rules are not in effect.
- ▶ You must have the write-down privilege to specify values for the security label columns, unless write-down rules are not in effect. If these rules are in effect and you do not have write-down privilege, DB2 assigns your security label as the value for the security label column for the rows that you are loading.

Ensure that any input data that is provided for a security label column is a valid security label. Security label columns are defined with the AS SECURITY LABEL clause.

If UNLOAD is to process a table space that contains a table that has multilevel security with row-level granularity, you must be identified to RACF and have an accessible valid security label. Each row is unloaded only if your security label dominates the data security label. If your security label does not dominate the data security label, the row is not unloaded, but DB2 does not issue an error message.

For more details about multi level security, refer to *DB2 Version 9.1 for z/OS Administration Guide*, SC18-9840

## 7.7 NOCOPYPEND option

Starting with DB2 9, you can use the NOCOPYPEND option to prevent a table space from being into the restrictive COPY pending state after LOADING without inline copy and with the LOG NO option. The NOCOPYPEND option should only be used when the data can be easily recreated from its original source. A typical example is a table for read-only applications, where the original source of the data is still available elsewhere, such as tables that summarize information from other tables.

Normally, when a table space is loaded with LOG NO and without inline copy (Example 7-10), the utility will end with RC=4 (warning messages) and the table space will be put in the COPYP state, as shown in Example 7-11. The table will be unavailable for insert, update or delete operations till a full image copy is taken afterwards.

*Example 7-10 Load LOG NO without inline copy*

---

```
LOAD DATA INDDN TSYSREC LOG NO REPLACE
EBCDIC CCSID(01148,00000,00000)
SORTKEYS 6243980
WORKDDN(TSYSUT1,TSORTOUT)
STATISTICS TABLE(ALL) INDEX(ALL KEYCARD)
INTO TABLE "S"."MTSTBP"
WHEN(00001:00002) = X'0003'
( "COD_DL_MAT"
  POSITION( 00003:00010) CHAR(00008)
.....
```

---

*Example 7-11 LOAD LOG NO without the inline copy job output*

---

```
.....
DSNU050I 278 13:51:21.13 DSNUGUTC - LOAD DATA INDDN TSYSREC LOG NO REPLACE EBCDIC CCSID(1148, 0, 0)
SORTKEYS 6243980 WORKDDN(TSYSUT1, TSORTOUT) STATISTICS TABLE(ALL) INDEX(ALL KEYCARD)
DSNU650I -DB9A 278 13:51:21.13 DSNURWI - INTO TABLE "S"."MTSTBP" WHEN(1:2)=X'0003'
DSNU650I -DB9A 278 13:51:21.13 DSNURWI - ("COD_DL_MAT" POSITION(3:10) CHAR(8),
.....
DSNU610I -DB9A 278 13:51:56.64 DSNUSUCD - SYSCOLDIST CATALOG UPDATE FOR S.I_MTSTBP_4 SUCCESSFUL
DSNU610I -DB9A 278 13:51:56.64 DSNUSUCD - SYSCOLDIST CATALOG UPDATE FOR S.I_MTSTBP_1 SUCCESSFUL
DSNU610I -DB9A 278 13:51:56.65 DSNUSUCD - SYSCOLDIST CATALOG UPDATE FOR S.I_MTSTBP_2 SUCCESSFUL
DSNU610I -DB9A 278 13:51:56.65 DSNUSUCD - SYSCOLDIST CATALOG UPDATE FOR S.I_MTSTBP_3 SUCCESSFUL
DSNU610I -DB9A 278 13:51:56.65 DSNUSUCD - SYSCOLDIST CATALOG UPDATE FOR S.I_MTSTBP_5 SUCCESSFUL
DSNU620I -DB9A 278 13:51:56.65 DSNUSEOF - RUNSTATS CATALOG TIMESTAMP = 2009-10-05-13.51.22.146212
DSNU381I -DB9A 278 13:51:56.76 DSNUGSRX - TABLESPACE MTSTBP.MTSTBP IS IN COPY PENDING
DSNU010I 278 13:51:56.76 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=4
```

---

However, when you add the NOCOPYPEND option to the LOAD control statement in Example 7-10, the utility will end with RC=0 (only informational messages) and the table space will no longer be left in the COPYP status, as shown in Example 7-12. The table space would be put in the INFORMATIONAL COPY PENDING state if records were discarded by the LOAD utility.

*Example 7-12 LOAD LOG NO NOCOPYPEND without inline copy job output*

---

```
.....
DSNU050I 278 13:53:06.18 DSNUGUTC - LOAD DATA INDDN TSYSREC LOG NO REPLACE EBCDIC CCSID(1148, 0, 0)
SORTKEYS 6243980 NOCOPYPEND WORKDDN(TSYSUT1, TSORTOUT) STATISTICS TABLE(ALL) INDEX(ALL KEYCARD)
DSNU650I -DB9A 278 13:53:06.18 DSNURWI - INTO TABLE "S"."MTSTBP" WHEN(1:2)=X'0003'
DSNU650I -DB9A 278 13:53:06.18 DSNURWI - ("COD_DL_MAT" POSITION(3:10) CHAR(8),
.....
DSNU610I -DB9A 278 13:53:41.12 DSNUSUCD - SYSCOLDIST CATALOG UPDATE FOR S.I_MTSTBP_4 SUCCESSFUL
DSNU610I -DB9A 278 13:53:41.12 DSNUSUCD - SYSCOLDIST CATALOG UPDATE FOR S.I_MTSTBP_1 SUCCESSFUL
DSNU610I -DB9A 278 13:53:41.13 DSNUSUCD - SYSCOLDIST CATALOG UPDATE FOR S.I_MTSTBP_2 SUCCESSFUL
DSNU610I -DB9A 278 13:53:41.13 DSNUSUCD - SYSCOLDIST CATALOG UPDATE FOR S.I_MTSTBP_3 SUCCESSFUL
DSNU610I -DB9A 278 13:53:41.14 DSNUSUCD - SYSCOLDIST CATALOG UPDATE FOR S.I_MTSTBP_5 SUCCESSFUL
DSNU620I -DB9A 278 13:53:41.14 DSNUSEOF - RUNSTATS CATALOG TIMESTAMP = 2009-10-05-13.53.07.200667
DSNU010I 278 13:53:41.31 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

---

DB2 will ignore the NOCOPYPEND specification if you also specify COPYDDN to make an inline image copy during the LOAD. As said before, specify the NOCOPYPEND option only if the data in the table space can be easily re-created by another LOAD job if the data is lost. If you do not take an image copy following the LOAD, you cannot recover the table space by using the RECOVER utility, and you might lose data

## 7.8 Loading NOT LOGGED table spaces

DB2 9 allows the creation of table spaces with the NOT LOGGED attribute. The NOT LOGGED attribute for a table space indicates that changes to tables in the table space are not recorded on the log.

The NOT LOGGED attribute can be used for data that can be easily recreated from its original source, so large MQTs could be candidates for NOT LOGGED table spaces. Table spaces use LOAD on a regular basis and the input sequential data sets are always available or easily recreated.

If you specify LOG YES on the LOAD statement, the data will not be logged when the table space has the NOT LOGGED attribute. If you specify LOG NO and the table space has the NOT LOGGED attribute, the table space will not be put in COPY PENDING, but in the INFORMATIONAL COPY PENDING (ICOPY) state, and the LOAD utility will end with RC=4 (warning). Refer to Example 7-13 for a LOAD of a NOT LOGGED segmented table space and Example 7-14 for a LOAD of a NOT LOGGED partitioned table space. If you also specify the NOCOPYPEND option, the table space will still be put in the ICOPY state, but the utility will now end with RC=0 (informational). This is shown in Example 7-15 on page 186.

*Example 7-13 LOAD of a NOT LOGGED segmented table space*

---

```

.....
DSNU050I    278 13:04:34.07 DSNUGUTC - LOAD DATA INDDN TSYSREC LOG NO REPLACE EBCDIC CCSID(1148, 0, 0)
SORTKEYS 6243980 WORKDDN(TSYSUT1, TSORTOUT) STATISTICS TABLE(ALL) INDEX(ALL KEYCARD)
DSNU650I    -DB9A 278 13:04:34.07 DSNURWI - INTO TABLE "S"."MTSTBP" WHEN(1:2)=X'0003'
DSNU650I    -DB9A 278 13:04:34.08 DSNURWI - ("COD_DL_MAT" POSITION(3:10) CHAR(8),
.....
DSNU620I    -DB9A 278 13:05:09.13 DSNUSEOF - RUNSTATS CATALOG TIMESTAMP = 2009-10-05-13.04.35.097302
DSNU568I    -DB9A 278 13:05:09.23 DSNUGSRX - TABLESPACE MTSTBP.MTSTBP IS IN INFORMATIONAL COPY PENDING STATE
DSNU010I    278 13:05:09.27 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=4

```

---

*Example 7-14 LOAD of a NOT LOGGED partitioned table space*

---

```

.....
DSNU050I    278 12:50:44.81 DSNUGUTC - LOAD DATA INDDN TSYSREC LOG NO REPLACE EBCDIC CCSID(1148, 0, 0) SORTKEYS 6243980
WORKDDN(TSYSUT1, TSORTOUT) STATISTICS TABLE(ALL) INDEX(ALL KEYCARD)
DSNU650I    -DB9A 278 12:50:44.82 DSNURWI - INTO TABLE "S"."MTSTBP" WHEN(1:2)=X'0003'
DSNU650I    -DB9A 278 12:50:44.84 DSNURWI - ("COD_DL_MAT" POSITION(3:10) CHAR(8),
.....
DSNU570I    -DB9A 278 12:51:36.32 DSNUGSRX - TABLESPACE MTSTBP.MTSTBP PARTITION 21 IS IN INFORMATIONAL COPY
PENDING
DSNU570I    -DB9A 278 12:51:36.32 DSNUGSRX - TABLESPACE MTSTBP.MTSTBP PARTITION 22 IS IN INFORMATIONAL COPY
PENDING
DSNU570I    -DB9A 278 12:51:36.32 DSNUGSRX - TABLESPACE MTSTBP.MTSTBP PARTITION 23 IS IN INFORMATIONAL COPY
PENDING
DSNU570I    -DB9A 278 12:51:36.32 DSNUGSRX - TABLESPACE MTSTBP.MTSTBP PARTITION 24 IS IN INFORMATIONAL COPY
PENDING
DSNU570I    -DB9A 278 12:51:36.32 DSNUGSRX - TABLESPACE MTSTBP.MTSTBP PARTITION 25 IS IN INFORMATIONAL COPY
PENDING
DSNU010I    278 12:51:36.51 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=4

```

---

*Example 7-15 LOAD of a NOT LOGGED partitioned table space with NOCOPYPEND option*

---

```
.....
DSNU050I   278 13:26:29.58 DSNUGUTC - LOAD DATA INDDN TSYSREC LOG NO REPLACE EBCDIC CCSID(1148, 0, 0)
SORTKEYS 6243980 NOCOPYPEND WORKDDN(TSYSUT1, TSORTOUT) STATISTICS TABLE(ALL) INDEX(ALL KEYCARD)
DSNU650I  -DB9A 278 13:26:29.59 DSNURWI - INTO TABLE "S"."MTSTBP" WHEN(1:2)=X'0003'
DSNU650I  -DB9A 278 13:26:29.59 DSNURWI - ("COD_DL_MAT" POSITION(3:10) CHAR(8),
.....
DSNU570I  -DB9A 278 13:27:20.52 DSNUGSRX - TABLESPACE MTSTBP.MTSTBP PARTITION 21 IS IN INFORMATIONAL COPY
PENDING
DSNU570I  -DB9A 278 13:27:20.52 DSNUGSRX - TABLESPACE MTSTBP.MTSTBP PARTITION 22 IS IN INFORMATIONAL COPY
PENDING
DSNU570I  -DB9A 278 13:27:20.52 DSNUGSRX - TABLESPACE MTSTBP.MTSTBP PARTITION 23 IS IN INFORMATIONAL COPY
PENDING
DSNU570I  -DB9A 278 13:27:20.52 DSNUGSRX - TABLESPACE MTSTBP.MTSTBP PARTITION 24 IS IN INFORMATIONAL COPY
PENDING
DSNU570I  -DB9A 278 13:27:20.52 DSNUGSRX - TABLESPACE MTSTBP.MTSTBP PARTITION 25 IS IN INFORMATIONAL COPY
PENDING
DSNU010I   278 13:27:20.70 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

---

You could also use NOT LOGGED table spaces with LOAD SHRLEVEL CHANGE without logging. Normally, you cannot specify LOG NO for LOAD RESUME YES SHRLEVEL CHANGE, but Online Load without logging can be accomplished by altering the table space to NOT LOGGED, running the Online Load, altering the table space back to LOGGED, and taking an image copy. However, unlike Offline LOAD RESUME, Online Load against a NOT LOGGED table space is not restartable if the LOAD fails. If an Online Load abends, and rollback is necessary for the NOT LOGGED table space, then the table space is placed in LPL/RECP status. An attempt to restart the LOAD results in a return code of 8 and message DSNU1154I. To recover from such a failure, the failed LOAD job must be terminated, the data must be recovered from a prior image copy, and the Online Load job rerun.

For more information about NOT LOGGED table spaces, refer to *DB2 9 for z/OS Technical Overview*, SG24-7330.

## 7.9 Unloading from a clone table

If you specify the CLONE keyword, DB2 will unload data from only clone tables in the specified table spaces. The UNLOAD utility will only process clone data if the CLONE keyword is explicitly specified. The use of CLONED YES in the LISTDEF statement is not sufficient. If you specify the name of the clone table in the FROM TABLE clause, you do not need to specify the CLONE keyword. The keyword CLONED YES or CLONED NO in a LISTDEF can be used to return only table spaces and index spaces that contain or do not contain cloned objects in the INCLUDE or EXCLUDE list.



## 7.10 Unloading from an image copy

Although this function already existed in DB2 V7, it is important to discuss considerations when unloading from image copies.

Due to regulations, users are now keeping image copies for a longer period of time, often 5 years or more. When the table and table space is recreated on another DB2 system and the UNLOAD utility is used, the user is accessing data in retained image copies that are no longer registered in SYSCOPY. The following can happen:

- ▶ The table space was a simple table space.  
Simple linear table spaces can no longer be created in DB2 9. In this case, the user should create a segmented table space, and UNLOAD can still process the image copy of the old simple table space. PK60612 (UK35132) has been modified to allow UNLOAD from an ICOPY of a table space that was non-segmented, even though now it is defined as segmented.
- ▶ The table column data types have been ALTERed since the image copy was taken.  
The versioning information is not available during UNLOAD. In this case, the user should keep the DDL that precisely matches each image copy.

Similar considerations apply when moving data using DSN1COPY

In Figure 7-1, we summarize the advantages of unloading from image copies:

- ▶ Unloading from an image copy does not interfere with the host table space and table. No locks are taken on table space and index spaces, thus avoiding lock contentions. The only reference is to the DB2 catalog for table definitions.
- ▶ The status of the table space does not affect the UNLOAD utility when unloaded from an image copy. The table space may be in STOP status or other restrict status.
- ▶ Either all columns of a table or a subset of columns of table can be unloaded using the FROM TABLE option. Data selection can be further qualified by the WHEN option and the SAMPLE option.

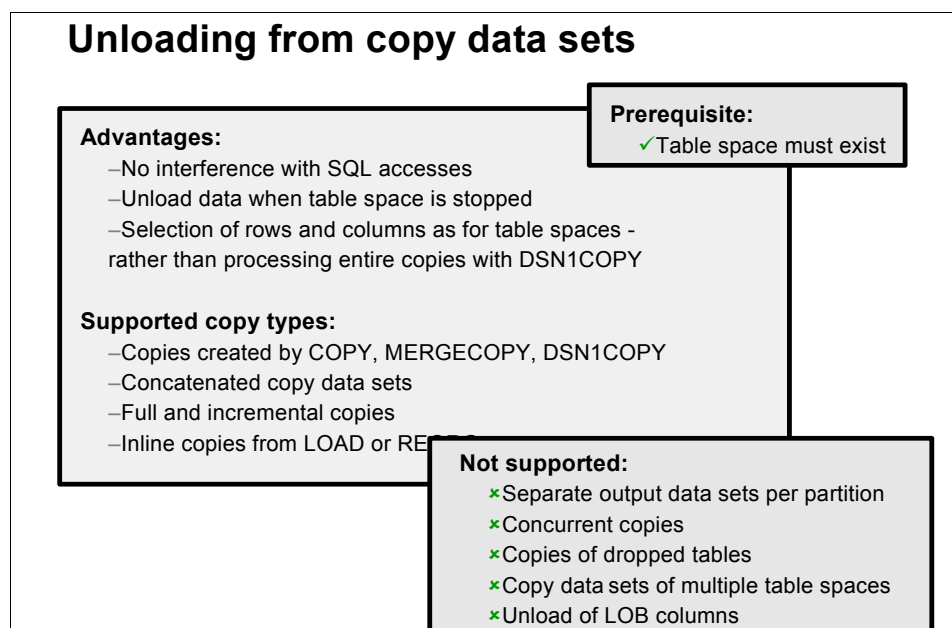


Figure 7-1 Unload from image copies

However, it is important to distinguish between the different types of image copies possible. If you want consistent data, unload from a full image copy taken with the SHRLEVEL REFERENCE. Such an image copy will only contain committed data. If you want to unload consistent data from different table spaces at the same point in time, use a common set of SHRLEVEL REFERENCE image copies as input (with the same START\_RBA in SYSIBM.SYSCOPY) created with a LISTDEF or LIST.

Remember that image copies taken with SHRLEVEL CHANGE can contain uncommitted rows. Inline copies can contain duplicate pages. If duplicate pages exist, the UNLOAD utility issues a warning message, and all the qualified rows in the duplicate pages will be unloaded into the output data set. Incremental image copies only contain partial data. If the data is compressed, the image copy must contain the compression dictionary or the unload will fail (see also 7.4, “Loading and unloading compressed data” on page 179). You cannot unload LOB or XML data from an image copy.

## 7.11 Loading and unloading decimal floating-point data

DECFLOAT is a new data type introduced in DB2 9 that lets you use decimal numbers with greater precision. DECFLOAT is similar to Packed Decimal in the sense that DECFLOAT processing deals with exact numbers, not numerical approximations of IEEE Floating Point.

A decimal floating-point value is a number with a decimal point. The position of the decimal point is stored in each decimal floating-point value. The maximum precision is 34 digits. The range of a decimal floating point number is either 16 or 34 digits of precision, and an exponent range of respectively  $10^{-383}$  to  $10^{+384}$  or  $10^{-6143}$  to  $10^{+6144}$ .

In addition to the finite numbers, decimal floating point numbers are able to represent one of the following named special values:

- ▶ Infinity: A value that represents a number whose magnitude is infinitely large (positive or negative).
- ▶ Quiet NaN: A value that represents undefined results and does not cause an invalid number condition. NaN is not a number.
- ▶ Signaling NaN: A value that represents undefined results that will cause an invalid number condition if used in any numerical operation.

When a number has one of these special values, its coefficient and exponent are undefined.

Both LOAD and UNLOAD are enhanced to handle the floating decimal data type. The DECFLOAT input data type is compatible with output data types SMALLINT, INTEGER, BIGINT, DECIMAL, FLOAT, and DECFLOAT. Input data types SMALLINT, INTEGER, BIGINT, DECIMAL, FLOAT, and DECFLOAT are compatible with the DECFLOAT output data type. Both LOAD and UNLOAD allow users to specify which mode of rounding is to be used. If DECFLOAT ROUNDMODE is not specified, the action is determined by the DSNHDECP DEF\_DECFLOAT\_ROUND\_MODE parameter.

The rounding modes supported are:

- ▶ ROUND\_CEILING: Round towards +infinity. If all of the discarded digits are zero or if the sign is negative, the result is unchanged other than the removal of discarded digits. Otherwise, the result coefficient should be incremented by 1 (rounded up).
- ▶ ROUND\_DOWN: Round towards 0 (truncation). The discarded digits are ignored.

- ▶ **ROUND\_FLOOR:** Round towards -infinity. If all of the discarded digits are zero or if the sign is positive, the result is unchanged other than the removal of discarded digits. Otherwise, the sign is negative and the result coefficient should be incremented by 1.
- ▶ **ROUND\_HALF\_DOWN:** Round to nearest. If equidistant, round down. If the discarded digits represent greater than half (0.5) of the value of a one in the next left position, then the result coefficient should be incremented by 1 (rounded up). Otherwise (the discarded digits are 0.5 or less), the discarded digits are ignored.
- ▶ **ROUND\_HALF\_EVEN:** Round to nearest. If equidistant, round so that the final digit is even. If the discarded digits represent greater than half (0.5) the value of a one in the next left position, then the result coefficient should be incremented by 1 (rounded up). If they represent less than half, then the result coefficient is not adjusted (that is, the discarded digits are ignored). Otherwise (they represent exactly half), and the result coefficient is unaltered if its right most digit is even, or incremented by 1 (rounded up) if its right-most digit is odd (to make an even digit).
- ▶ **ROUND\_HALF\_UP:** Round to nearest. If equidistant, round up. If the discarded digits represent greater than or equal to half (0.5) of the value of a one in the next left position, then the result coefficient should be incremented by 1 (rounded up). Otherwise, the discarded digits are ignored.
- ▶ **ROUND\_UP:** Round away from 0. If all of the discarded digits are zero the result is unchanged other than the removal of discarded digits. Otherwise, the result coefficient should be incremented by 1 (rounded up).

## 7.12 Loading and unloading LOB and XML data

Depending on the way the LOB or XML data is provided, there are three ways the LOAD utility can be used to load LOB or XML data:

- ▶ Loading LOB or XML data as normal data fields from the LOAD input file
- ▶ Using file reference variables when each LOB value or XML document is stored in a separate input file
- ▶ Using the cross loader (LOB only)

In DB2 9, a new SORTKEYS NO keyword has been introduced to allow restartability with the RESTART(CURRENT) option of LOAD jobs with LOB data.

There are two ways to unload LOB or XML data with the UNLOAD utility:

- ▶ Unloading LOB or XML data as normal data fields in the UNLOAD output file.
- ▶ Using file reference variables to unload each LOB value or XML document into a separate file. The name of this file is then stored in the normal UNLOAD output file.

**Consideration:** You cannot unload LOB data or XML data from image copies.

In Figure 7-2, you can see the architecture of LOB data where the actual LOB data is stored in a separate LOB table space, linked to the base table using the auxiliary index. For XML, the XML documents will be stored in a separate implicitly created XML table space and the XML documents are linked via the document ID index

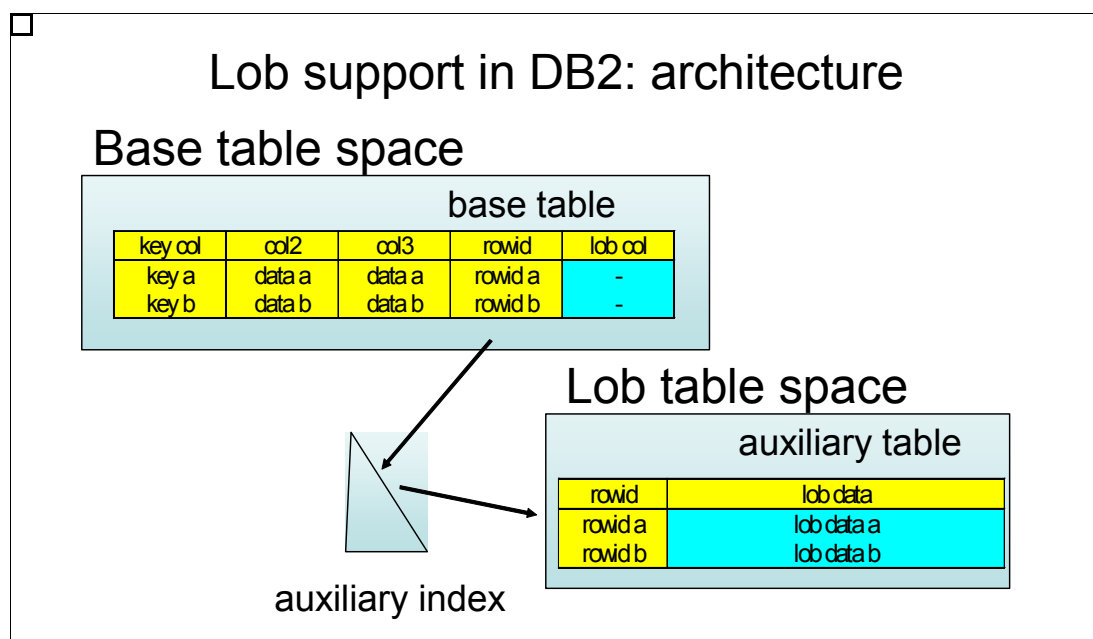


Figure 7-2 LOB architecture

For examples and other considerations about loading and unloading LOB data, refer to *LOBs with DB2 for z/OS: Stronger and Faster*, SG24-72700.

### 7.12.1 Loading and unloading LOB or XML data as normal data columns

This method can be used when the LOB or XML values are stored in the normal LOAD input file or are stored in the normal UNLOAD output file together with the other data fields of the base table. Because the maximum record length of a sequential file in z/OS is 32 KB, this method can only be used to LOAD or UNLOAD LOB columns or XML documents of 32 KB or less. The sum of the length of all normal data fields and LOB/XML fields in the file cannot exceed 32 KB. This method is only used if the LOBs or XMLs have an actual length smaller than 32 KB. The use of spanned records in a sequential data set is not currently supported in DB2.

LOB fields are vary in length and use a valid 4-byte binary length field preceding the data. Specify CLOB, BLOB, or DBCLOB in the field specification portion of the LOAD or UNLOAD statement. XML fields vary in length and use a valid 2-byte binary length field preceding the data. It is also possible to LOAD or UNLOAD LOB or XML fields from and to a delimited file, where the fields are delimited by a string delimiter and separated from other fields by column delimiters.

You can use the PRESERVE WHITESPACE option to preserve the white spaces when loading a XML document. The default is not to preserve white spaces.

DB2 does not compress an XML table space during the LOAD process. If the XML table space is defined with COMPRESS YES, the XML table space is compressed during REORG.

If you perform a LOAD operation on a base table that contains a LOB or XML column, DB2 does not collect inline statistics for the related LOB and XML table space or its indexes.

Normally, the provided input file for LOAD will be the result of:

- ▶ UNLOAD utility
- ▶ DSNTIAUL utility
- ▶ Generated by another application

## 7.12.2 Loading and unloading LOB or XML data using file reference variables

This method is used when the LOB or XML values are stored in separate input files or are stored in separate output files. The normal input file contains the data for the non-LOB and non-XML columns of the base table and the names of the LOB and XML files. The normal output file will contain the data for the non-LOB and non-XML columns of the base table and the names of the LOB and XML files. The sum of the length of all normal data fields and the LOB and XML file names cannot exceed 32 KB.

The LOB and XML input files for LOAD can be any of these types:

- ▶ A sequential file
- ▶ A member of a PDS or PDSE
- ▶ A HFS file on a HFS directory

The LOB and XML input files contain the entire LOB value or XML document and the name of this file is stored in the normal load input file as a CHAR or VARCHAR field. So, instead of containing the whole LOB or XML value, the normal input file now only contains a file name, which in most cases no longer causes the sequential file to hit the 32 KB limit. This is shown in Figure 7-3 for LOBs stored in a PDS and in Figure 7-4 on page 192 for LOBs stored in a HFS directory.

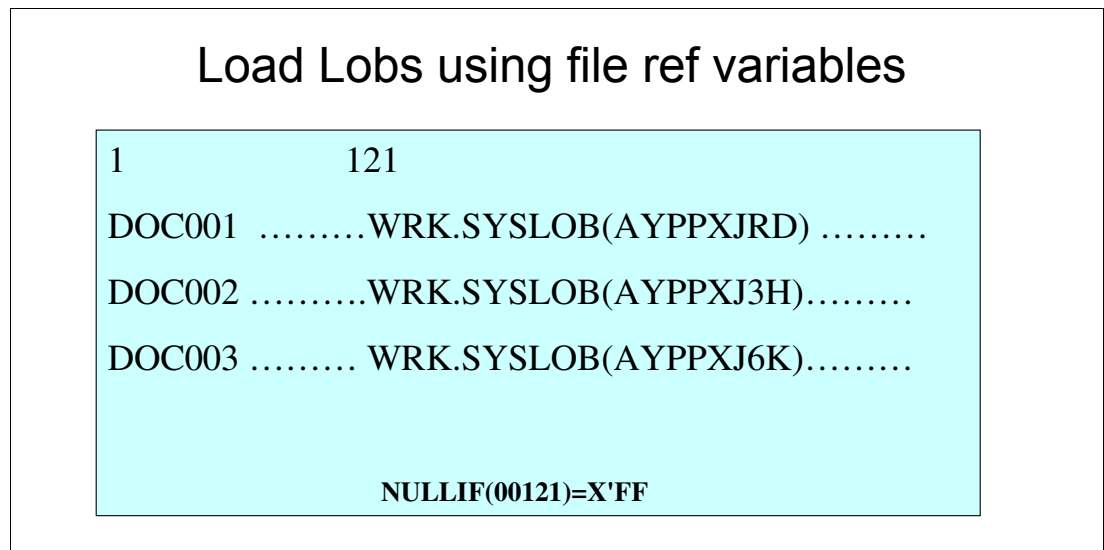


Figure 7-3 File references for LOBs stored in a PDS or PDSE

## Load LOBs using file ref variables

```
1          121
DOC001 ...../u/davy/AYPPXJRD .....
DOC002 ...../u/davy/AYPPXJ3H.....
DOC003 ..... /u/davy/AYPPXJ6K.....

      NULLIF(00121)=X'FF
```

Figure 7-4 File references for LOBs stored on a HFS directory

Additional keywords have been added to the CHAR and VARCHAR field specifications of the LOAD utility to support a file name as the input for the actual LOB value:

- ▶ BLOBF: The input field contains the name of a file with a BLOB value or XML document.
- ▶ CLOBF: The input field contains the name of a file with a CLOB value or XML document.
- ▶ DBCLOBF: The input field contains the name of a file with a DBCLOB value or XML document.

In the case of CLOBF and DBCLOBF, CCSID conversions are done when the CCSID of the input data is different than the CCSID of the table space. (EBCDIC, ASCII, UNICODE, or CCSID keywords might have been specified for the input data; the default is EBCDIC input data.) In case of BLOBF, no conversions are done (binary format). When the input field of a BLOBF, CLOBF, or DBCLOBF is NULL, the resulting LOB value or XML value is NULL (null indicator field for the CHAR or VARCHAR field specified in the NULLIF keyword is hex FF).

The provided input file for LOAD is the result of:

- ▶ UNLOAD utility
- ▶ DSNTIAUL utility
- ▶ Generated by another application

Similarly, the LOB and XML output files from UNLOAD can be any of these types:

- ▶ A member of a PDS or PDSE. The member names will be automatically generated.
- ▶ A HFS file on a HFS directory. The data set names in the directory will be automatically generated.

For UNLOAD, the actual files to be created or used for storing the LOB or XML values generated from a TEMPLATE definition with a name to be specified along with the BLOBF, CLOBF, or DBCLOBF keywords. The TEMPLATE definition is used to specify the characteristics of the LOB or XML unload files (DSNTYPE PDS, LIBRARY, or HFS). The BLOBF keyword implies that the XML document is to be unloaded in binary XML format.

The member names that are generated for a PDS or PDSE or the data set names for a HFS directory have a length of 8 and are uniquely generated from the system clock. They have the same look as when using the &UNIQ. or &UQ. variable in a TEMPLATE definition (refer to Figure 7-3 on page 191 and Figure 7-4 on page 192).

You cannot unload a LOB object or XML document to a sequential file. Use the sample program DSNTIAUL instead.

When using load and unload with file reference variables, all file access time appears as an “other service task” class 3 suspension. CPU time is accumulated as Database System Server TCB time.

There are also performance improvements in DB2 9 compared to DB2 V8 when loading or unloading LOBs using file reference variables:

- ▶ DB2 V8: For every LOB, DB2 uses SVC99 (dynamic allocation/deallocation) and SVC19 (Open) and SVC20 (Close). When using partitioned data sets, SVC 19 uses SVC18 (FIND) for Load and SVC 21 (STOW) for Unload.
- ▶ DB2 9: DB2 calls UNIX System Services directly. DB2 9 avoids all SVCs for UNIX System Services file systems. Moreover, DB2 9 (PK75216) keeps PDS/PDSE open and uses BPAM to manage the directory. Load uses DESERV to scan the directory sequentially. Unload uses STOW to update the directory for each LOB.

In DB2 9, elapsed time reductions of four times have been measured. HFS and zFS are faster than PDS/PDSE (especially for Unload, not so much for Load). See Figure 7-5.

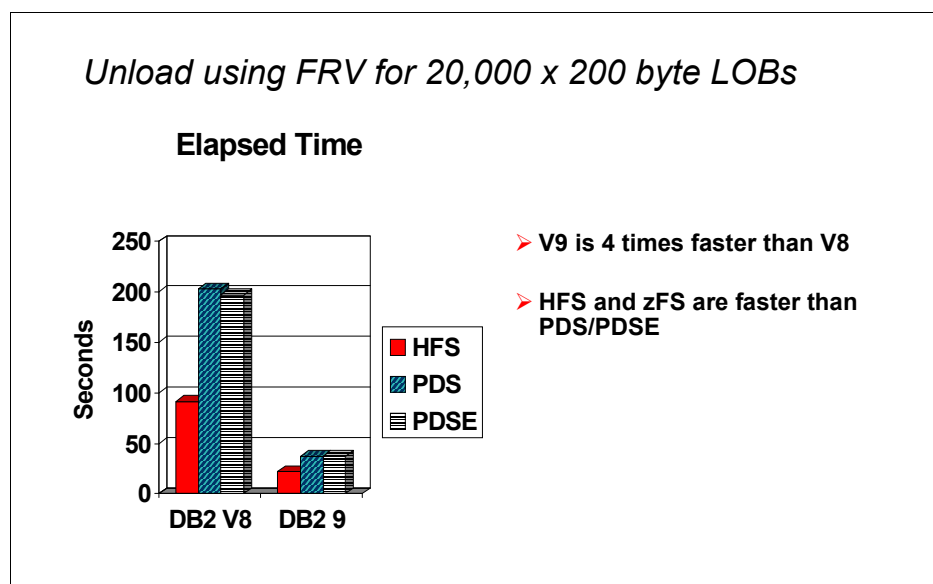


Figure 7-5 Unloading LOBs

### 7.12.3 Loading LOB data with the cross loader

The cross loader function of the LOAD utility can be used to load LOB data that resides in another table on the same or another location connected through DRDA. When you use the cross loader function for LOBs greater than 32 KB, DB2 uses a separate buffer for LOB data and only stores 8 bytes per LOB column in the cursor result set buffer.

The sum of the lengths of the non-LOB columns plus the sum of 8 bytes per LOB column cannot exceed 32 KB. The separate LOB buffer resides in storage above the 16 MB line and is only limited by the available memory above the 16 MB line. The message DSNU1178I is issued when:

- The sum of the lengths of all non-LOB columns + 8 bytes per LOB column exceeds 32 KB.
- The sum of the lengths of all LOB columns exceeds half of the above-the-line available memory.

If a user gets message DSNU1178I, increasing the region size likely results in successful execution of the LOAD utility.

If a table contains LOB columns, there is at least one explicit or hidden ROWID column. If this ROWID column in the target table is not hidden and created with the GENERATED ALWAYS clause (the recommended default), you cannot specify this column in the select list of the cursor. Instead, DB2 generates the target ROWID value during loading. Do not forget to create a unique index on the ROWID column if you specified GENERATED BY DEFAULT or you fail with error message DSNU309I NOT ALL REQUIRED UNIQUE INDEXES HAVE BEEN DEFINED FOR TABLE *tablename*.

Loading XML data using the cross loader is currently not supported.

## 7.12.4 Loading LOB data with SORTKEYS NO

When a LOAD SHRLEVEL NONE of a table with a LOB column fails, it can only be restarted with RESTART(CURRENT). This restriction was introduced with LOB support. However, when Parallel Index Build (PIB) is used (the default since DB2 V8), the LOAD that abends can only be restarted with RESTART(PHASE). To resolve this incompatibility issue, in DB2 9, you can now specify SORTKEYS NO to disable the PIB and to allow the LOAD SHRLEVEL NONE to restart with RESTART(CURRENT).

In DB2 9, DB2 will also allow RESTART(PHASE) for LOAD REPLACE of a table with LOB columns.

For a load of a table with LOB columns, using LOAD REPLACE SHRLEVEL NONE, the SORTKEYS option will determine how the utility can be restarted. If you specify SORTKEYS NO, you can restart with either RESTART(CURRENT) or RESTART(PHASE). If you do not specify SORTKEYS NO, you can restart only with RESTART(PHASE).

For a LOAD RESUME YES SHRLEVEL NONE of a table with LOB columns to be re-startable, SORTKEYS NO must be specified. In this case, you will be able to perform a RESTART(CURRENT).

Use the SORTKEYS NO option only when loading LOB data and when you want your job to be re-startable, because it will negatively influence the performance because of the parallelism being disabled.



## 7.13 Cross loader considerations

Although the cross loader functionality, which allows you to specify the input to the LOAD utility as the result set of an SQL query launched against tables on a local or remote DB2 system, was already available in DB2 V7, we would like to repeat here some elements you should take into consideration when using it.

The data is read from the source location with a dynamic SQL statement and loaded in a table at the target location by the LOAD utility. It is a single job process that replaces the typical sequence of jobs of unloading, file transfer, and loading the data. The source data can be on the local system, on a remote DRDA server, or on any system accessible via IBM InfoSphere Classic Federation Server.

A typical cross loader example consists of the definition of the dynamic SQL statement via the EXEC SQL DECLARE CURSOR utility statement, followed by a LOAD utility statement referring to this cursor. This is an example of the syntax:

```
EXEC SQL
    DECLARE c1 CURSOR FOR
        select query
ENDEXEC
LOAD DATA
    INCURSOR c1
    INTO TABLE creator.table
```

In Example 7-16, we load an existing summary table called EMPSUMMARY with data coming from the local employee table EMPLOYEE. The aggregation is done in the SQL statement of the CURSOR definition.

*Example 7-16 Simple cross loader example*

---

```
EXEC SQL
DECLARE C1 CURSOR FOR
SELECT JOB,MAX(SALARY) AS MAX_SAL,MIN(SALARY) AS MIN_SAL
FROM EMPLOYEE
GROUP BY JOB
ENDEXEC
LOAD DATA REPLACE
INCURSOR C1
INTO TABLE EMPSUMMARY
```

---

The following rules apply to the cursor you declare in the EXEC SQL statement:

- ▶ You must always declare the cursor before the LOAD statement and the cursor name can only be declared once within the whole utility input stream. It can be referred in multiple LOAD statements for LOADING different target tables with the same data. It can also be reused in one LOAD statement containing multiple INTO TABLE clauses to LOAD the same data in different target tables (in the same table space).
- ▶ The table being loaded cannot be part of the select statement. You cannot load into the same table where you defined the cursor.
- ▶ If a cursor is used more than once in the utility job step, the data is transferred more than once as well. Each time you refer to a cursor name in a LOAD statement, the SQL statement is re-executed. There is no buffering or reuse of the previous transferred data. The result sets can differ if the source data is being updated or if you use time dependent functions.

- ▶ The column names in the result set of the select statement must be identical to the column names in the table being loaded. This can be achieved by using the AS clause in the SQL statement. Pay special attention to derived column names that are the result of column functions such as SUM or AVG.
- ▶ You are able to skip unwanted columns in the result set with the LOAD IGNOREFIELDS YES option, which skips any columns in the cursor result set that are not present in the target table being loaded.
- ▶ The sequence of the columns in the result set may differ from the sequence of the columns in the table being loaded. DB2 matches the columns by their names and not by their sequence.
- ▶ The number of columns in the cursor can be less than the number of columns in the target table. All missing columns are loaded with their default values. If the data types in the target table do not match with the data types in the cursor, DB2 tries to convert as much as possible between compatible data types. You might use DB2 supplied built-in functions or your own developed UDFs in the SQL statement to force more sophisticated conversions. An example is the CHAR function, which allows you to convert from INTEGER to CHARACTER. If the encoding scheme of the source data in the cursor and the target table differ, DB2 automatically converts the encoding schemes. An example may be conversion from EBCDIC to UNICODE or from ASCII to EBCDIC.
- ▶ As already explained in 7.12.3, “Loading LOB data with the cross loader” on page 193, you can transfer entire LOB columns. However, you cannot transfer XML columns.

Apart from these rules, the SQL statement in the declare cursor definition can be any valid SQL statement, including joins, unions, conversions, aggregations, special registers, and UDFs. Remote tables are always referred by their three-part-name LOCATION.CREATOR.NAME or by an ALIAS name CREATOR.NAME. You cannot use an explicit CONNECT statement to connect to the remote location.

The cross loader uses package DSNUGSQL in collection DSNUTIL, which must be bound locally and at the remote DRDA servers from which you want to transfer data. It uses the default system utility plan DSNUTIL.

You can use any option of the LOAD utility except the following ones:

- ▶ SHRLEVEL CHANGE: There is no support for online LOAD in the Cross Loader.
- ▶ FORMAT: You cannot specify the UNLOAD or SQL/DS formats.
- ▶ FLOAT(IEEE): The cursor always returns FLOAT(S390).
- ▶ EBCDIC,ASCII,UNICODE: The cross loader always automatically converts the encoding schemes. The target table must be created with the correct CCSID.
- ▶ NOSUBS: You must accept substitution characters in a string.
- ▶ CONTINUEIF: You cannot treat each input record as a part of a larger record.
- ▶ WHEN: You cannot use the WHEN option to filter the result set of the cursor. Instead, filter the result set by using the appropriate WHERE clauses in the select statement.
- ▶ Field specifications: You cannot specify field specifications in the LOAD Statement.

We recommend that you load your data in clustering sequence. If loading from a sequential data set, this can be done by first sorting the sequential data set in clustering sequence. With the cross loader, this can be achieved by sorting the result set of the cursor by using an ORDER BY statement on the clustering columns.

You can load partitions in parallel by specifying a different cursor for each partition.

Because DB2 does not do space estimations for the cursor result set, specify a large enough SORTKEYS value to enable Parallel Index Build and provide SPACE estimations to the sort and inline copy data set templates. Refer to 4.4, “Parallelism with the LOAD utility” on page 82. In Example 7-17, we also sort the result set in a clustering sequence before loading.

*Example 7-17 Specifying SORTKEYS and SPACE estimations to the cross loader*

---

```

TEMPLATE TSORTOUT DSN('&US..&SS..&DB..&SN..S&JU(3,5)..#&TI.')
                SPACE(5000,1000) TRK MAXPRIME 65536
                DISP(MOD,DELETE,CATLG)
TEMPLATE TSYSUT1  DSN('&US..&SS..&DB..&SN..U&JU(3,5)..#&TI.')
                SPACE(5000,1000) TRK MAXPRIME 65536
                DISP(MOD,DELETE,CATLG)
TEMPLATE TSYSCOPY
                DSN('&US..&SS..&DB..&SN..&IC.&JU(3,5)..#&TI.')
                SPACE(5000,1000) TRK MAXPRIME 65536
                DISP(MOD,CATLG,CATLG)
EXEC SQL
  DECLARE C1 CURSOR FOR SELECT * FROM "S"."MTSTBX"
  ORDER BY COD_DL_MAT,NR_DL_MAT
ENDEXEC
LOAD DATA INCURSOR C1 LOG NO REPLACE
  SORTKEYS 6243980 SORTDEVT 3390
  COPYDDN(TSYSCOPY) WORKDDN(TSYSUT1,TSORTOUT)
  STATISTICS TABLE(ALL) INDEX(ALL KEYCARD)
  INTO TABLE "S"."MTSTBO"
  SHRLEVEL NONE
/*

```

---

## 7.14 LOAD SHRLEVEL CHANGE and clone tables

The LOAD SHRLEVEL CHANGE capability was introduced in DB2 V7 and allows you to load data with the LOAD utility, while having concurrent SQL access on the same table space, as shown in Figure 7-6.

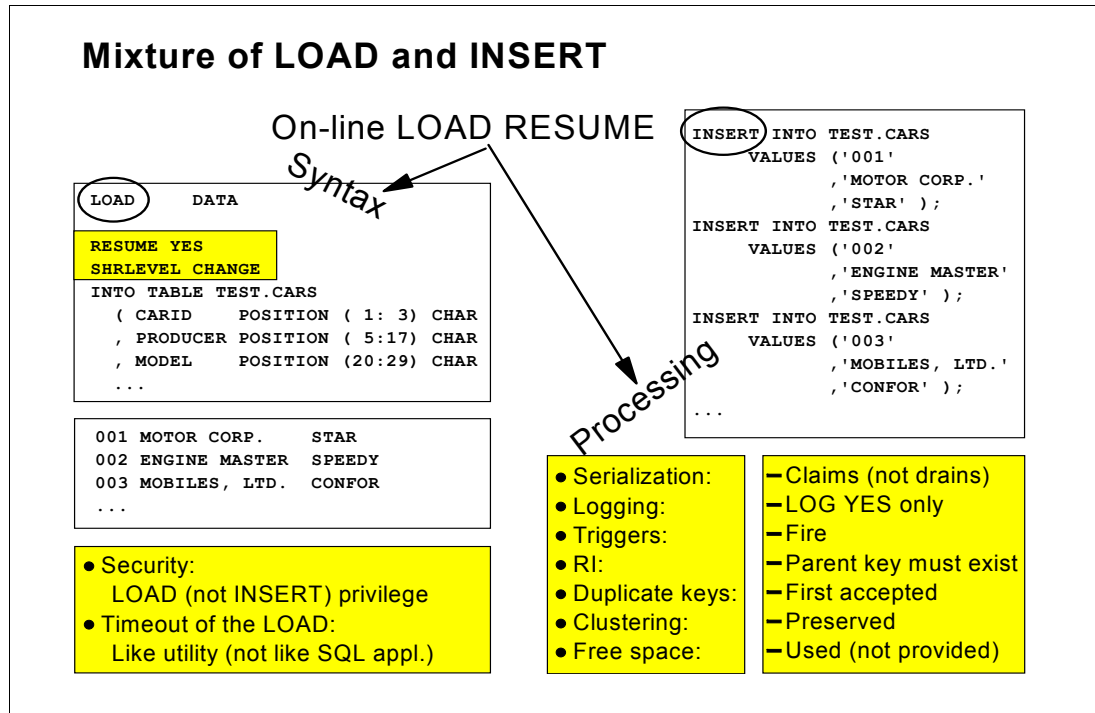


Figure 7-6 REORG SHRLEVEL CHANGE

LOAD SHRLEVEL NONE drains all access to the table space which prevents the data from being available for SQL applications. With LOAD SHRLEVEL CHANGE, there is no need anymore to write your own INSERT programs for loading data, thus avoiding the drain and allowing concurrent SQL access.

LOAD SHRLEVEL CHANGE behaves externally as a LOAD, but it works like a mass INSERT. Triggers are fired and the index building, duplicate key, and referential constraint checking will be handled by SQL insert processing. The records that fail during the insertion are written to the discard data set.

LOAD SHRLEVEL CHANGE always requires RESUME YES and LOG YES. So LOAD REPLACE SHRLEVEL CHANGE is not allowed because it is basically an INSERT processing. You could use NOT LOGGED table spaces to avoid the logging. Refer to 7.8, "Loading NOT LOGGED table spaces" on page 185 for more information.

If you need LOAD REPLACE SHRLEVEL CHANGE capabilities, use a clone table with LOAD REPLACE SHRLEVEL NONE and switch the clone table afterwards. This will behave as a LOAD REPLACE SHRLEVEL CHANGE with virtually no outage.

Clone tables were introduced in DB2 9 and provide you with the ability to generate a table with the exact same attributes as a base table that already exists. The clone table is created in the same table space as the base table. Once the clone table is created, you can independently work with it (you can, for example, load it or insert rows into it).

The clone table is structurally identical to the base table in every way (that is, it has the same number of columns, column names, data types, check constraints, and so on), and it is also created with the same indexes, before triggers, LOB objects, and so on. DB2 creates the clone objects automatically when it creates the clone table. Although the clone table itself has a different name and might have a different schema associated to it, the clone table objects are referred to by the same names as those that are used for the base table objects (that is, the base and clone table share all object descriptions).

One important restriction is that the base table must be created in a universal table space before you can create a clone table. So you might need to migrate your existing base table to a UTS first before creating the clone table. Both partition by growth or range partitioned table spaces are possible. Other restrictions are that the base table cannot contain after triggers and may not be involved in referential constraints. It cannot be an MQT.

For more information about how to create a clone table, populate it using LOAD or INSERT, and switch it with the base table using the EXCHANGE DATA SQL statement, refer to *DB2 9 for z/OS Technical Overview*, SG24-7330.

## 7.15 Skip locked data during UNLOAD

The SKIP LOCKED DATA option was introduced in DB2 9 to allow a transaction to skip rows that are incompatibly locked by other transactions. Because the SKIP LOCKED DATA option skips these rows, the performance of some applications can be improved by eliminating lock wait time. However, it should only be used by applications that can reasonably tolerate the absence of the skipped rows in the returned data. The SKIP LOCKED DATA option can also be used with the UNLOAD utility.

When unloading from a table space, you can specify the SHRLEVEL concurrency option for the UNLOAD utility:

- ▶ **SHRLEVEL REFERENCE:** Specifies that during the unload operation, rows of the tables can be read, but cannot be inserted, updated, or deleted by other DB2 threads. When you specify SHRLEVEL REFERENCE, the UNLOAD utility drains writers on the table space from which the data is to be unloaded in the UTILINIT phase. When data is unloaded from multiple partitions, the drain lock is obtained for all of the selected partitions.
- ▶ **SHRLEVEL CHANGE:** Specifies that rows can be read, inserted, updated, and deleted from the table space or partition while the data is being unloaded. Two isolation levels can be specified:
  - **ISOLATION UR:** Indicates that uncommitted rows, if they exist, will also be unloaded. The unload operation is performed with minimal interference from the other DB2 operations that are applied to the objects from which the data is being unloaded.
  - **ISOLATION CS:** Indicates that the UNLOAD utility is to read rows in cursor stability mode. With CS, the UNLOAD utility assumes CURRENTDATA(NO).

The SKIPPED LOCKED DATA option can only be specified with UNLOAD SHRLEVEL CHANGE ISOLATION CS. When specified, UNLOAD will skip rows on which incompatible locks are held by other transactions. This option applies to row level or page level lock.

## 7.16 Loading and unloading HFS files

With the introduction of file reference variables for loading and unloading LOB objects, it became possible to load and unload LOB objects from and to HFS files on an HFS directory (refer to 7.12.2, “Loading and unloading LOB or XML data using file reference variables” on page 191 for more information).

APAR PK70269 also introduces the capability to load and unload regular data from and to an HFS file. Therefore, a new PATH keyword was introduced that can be used in the TEMPLATE definition of INDDN for LOAD and UNLDDN for UNLOAD.

The following rules apply to the PATH keyword:

- ▶ It can only be used for the actual data to be loaded (INDDN) or unloaded (UNLDDN), not for other files, such as the PUNCHDDN for UNLOAD.
- ▶ You cannot use name substitution variables like &SS., &DB., or &SN.
- ▶ HFS directories and files are case sensitive, so respect the uppercase and lowercase.
- ▶ You can specify the content type of the file (TEXT, BINARY, or RECORD). The default value is RECORD.
- ▶ You can specify the record format (RECFM) and record length (LRECL) of the file.
- ▶ You can specify optional read-write and waiting options by using the PATHOPTS keyword.
- ▶ You can specify optional security options (permissions) by using the PATHMODE keyword.
- ▶ You can specify disposition options KEEP or DELETE for the file when the utility ends normal or abnormal by using the FILEKEEP keyword.
- ▶ See the APAR description of APAR PK70269 for more details.

In Example 7-18, we show how to use the PATH keyword to unload a DB2 table into an HFS file.

*Example 7-18 Using the PATH keyword to unload DB2 data into an HFS file*

---

```
TEMPLATE TSYPUN DSN('&US.&SS.&DB.&SN..P&JU(3,5)..PUNCH')
          DISP(MOD,CATLG,CATLG)
TEMPLATE TSYSREC PATH /u/db2r6/mtstbo_sysrec01
          PATHDISP(KEEP,DELETE)
UNLOAD DATA FROM TABLE S.MTSTBO
UNLDDN(TSYSREC) PUNCHDDN(TSYPUN)
/*
```

---

As a result, the HFS file /u/db2r6/mtstbo\_sysrec01 is created with attributes shown in Example 7-19.

*Example 7-19 Attributes of the HFS unload file*

---

```

Edit  Help
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
      Display File Attributes

Pathname : /u/db2r6/mtstbo_sysrec01
More:      +
File type . . . . . : Regular file
Permissions . . . . . : 600 rw-----
Access control list . : 0
File size . . . . . : 298462244
File owner . . . . . : DB2R6(1128)
Group owner . . . . . : SYS1(2)
Last modified . . . . : 2009-10-14 18:06:54
Last changed . . . . . : 2009-10-14 18:06:54
Last accessed . . . . . : 2009-10-14 18:06:32
Created . . . . . : 2009-10-14 18:06:32
Link count . . . . . : 1
F1=Help      F3=Exit      F4=Name
F7=Backward  F8=Forward   F12=Cancel

```

---

We can load the same file back into DB2, as shown in Example 7-20.

*Example 7-20 Using the PATH keyword to load a HFS file to DB2*

---

```

TEMPLATE TSYSREC
    PATH('/u/db2r6/mtstbo_sysrec01')
    PATHDISP(KEEP,KEEP)
TEMPLATE TSORTOUT DSN('&US..&SS..&DB..&SN..S&JU(3,5)..#&TI.')
    SPACE(1000,1000) TRK MAXPRIME 65536
    DISP(MOD,DELETE,CATLG)
TEMPLATE TSYSUT1 DSN('&US..&SS..&DB..&SN..U&JU(3,5)..#&TI.')
    SPACE(1000,1000) TRK MAXPRIME 65536
    DISP(MOD,DELETE,CATLG)
TEMPLATE TSYSCOPY
    SPACE(5000,1000) TRK MAXPRIME 65536
    DSN('&US..&SS..&DB..&SN..&IC.&JU(3,5)..#&TI.')
    DISP(MOD,CATLG,CATLG)
LOAD DATA INDDN(TSYSREC) LOG NO REPLACE
SORTKEYS 6243980
EBCDIC CCSID(00037,00000,00000)
COPYDDN(TSYSCOPY) WORKDDN(TSYSUT1,TSORTOUT)
SORTDEVT 3390
STATISTICS TABLE(ALL) INDEX(ALL KEYCARD)
INTO TABLE "S"."MTSTBO"
WHEN(00001:00002) = X'0003'
( "COD_DL_MAT"
    POSITION( 00003:00010) CHAR(00008)
, "NR_DL_MAT"
    POSITION( 00011:00036) TIMESTAMP EXTERNAL(026)
, "TY_DL_MAT"
.....

```

---

Because DB2 does not take into account the size of the input HFS data set, SORTKEYS are specified to enable the PIB, and space parameters are provided to the other templates to prevent the DSNU1036I message (UNABLE TO ESTIMATE SPACE REQUIREMENTS).

## 7.17 Loading and unloading virtual files

Here we distinguish between two types of virtual files: BatchPipes® and UNIX System Services pipes

- ▶ **BatchPipes:** These virtual files can be used in a batch job to transfer data from one job to another concurrently. Consider the common case of a batch job that writes a new sequential data set, followed by another job that reads it. Traditionally, the second job (the “reader”) cannot start processing the data set until the first job (the “writer”) has finished writing to it. So the two jobs cannot run concurrently. With BatchPipes, a short in-memory queue is created. This queue (or pipe) is written to by the writer job and read from by the reader job. As records become available in the pipe, the reader can process them, removing them from the front of the queue. The writer places records on the back of the queue. Readers and writers for an individual pipe use a slightly modified DD statement. Often the only change is to add “SUBSYS=ssnm” to the DD statement to specify the pipe name. At a minimum, all jobs using the pipe must specify the same pipe name, LRECL, and RECFM parameter by using the DSN parameter on the DD statement. Also, FTP can use Batch Pipes for put and get requests.
- ▶ The BatchPipes approach has a number of advantages:
  - The reader and writer must run at the same time, increasing concurrency and shortening their combined run time.
  - Whereas the traditional approach would have involved physical I/O, the BatchPipes approach is “in memory”. This reduction in I/O has the potential to reduce run time.
  - There is no disk or tape space requirement for the pipe, so this approach is essentially limitless.
- ▶ **UNIX System Services Pipes:** A similar approach is possible in the UNIX System Services environment. Pipes are very commonly used in UNIX environments. One UNIX System Services process can write in a pipe and another process is able to read from it simultaneously. It is also possible to set up a UNIX pipe between a UNIX System Services process running on one LPAR and a UNIX System Services process on another LPAR, or between a UNIX System Services process on one LPAR and a Linux on System z process running on another LPAR, using HiperSockets™ for the TCP/IP communication. UNIX processes on the mainframe can use pipes to interchange data with UNIX processes on other UNIX platforms or Windows platforms.

BatchPipes support was introduced in DB2 with APAR PK34251. It allows DB2 to specify a BatchPipe as input to the LOAD utility. The Template definition for the INDDN input data set now allows you to specify the SUBSYS, LRECL, and RECFM parameters of a MVS batch pipe. The LOAD job can run as a batch job or a stored procedure (DSNUTILS). The LOAD utility will end if the writer closes the pipe or when the FTP process ends. Because DB2 is unable to read backwards in a pipe, if the LOAD job fails, it cannot be restarted. Do a terminate utility in that case and reschedule the LOAD job. For the same reason, discard processing is not supported.

**Important:** if you want to load from a BatchPipe, always use the template definition. Do not allocate the file in the JCL with a DD statement.



In Figure 7-7, we show how a DataStage® job running on Linux on System z creates an output that is sent through FTP and the BatchPipes extension to the BatchPipes subsystem on z/OS. While transferring data to the BatchPipe, DataStage invokes the DSNUTILS administration stored procedure to trigger the LOAD utility. LOAD reads from the BatchPipe and loads the data into the desired DB2 for z/OS table. Because creating the data, transferring the data, and the LOAD occur simultaneously, the elapsed time of the whole process is much shorter than doing these tasks serially.

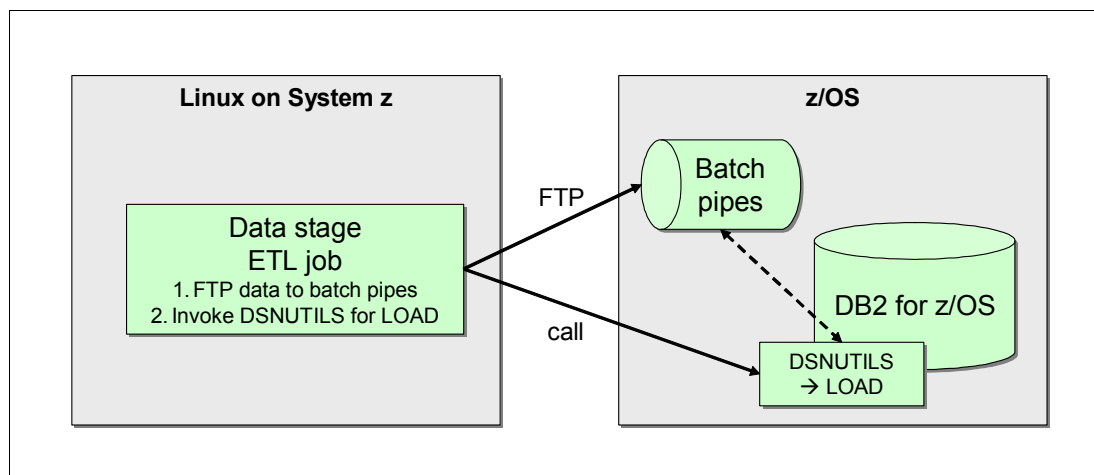


Figure 7-7 Interacting with BatchPipes

Refer to *Enterprise Data Warehousing with DB2 9 for z/OS*, SG24-7637 for more information about DataStage.

UNIX System Services Pipes support was introduced in DB2 with APAR PK70269, the same APAR that introduced the HFS support, as explained in 7.16, “Loading and unloading HFS files” on page 200. It allows you to specify additional parameters for UNIX System Services pipes in the PATH keyword. You can direct the LOAD utility to wait or not to wait until another process opens the pipe for writing. UNLOAD to a UNIX System Services pipe is supported.

The main differences between using BatchPipes and UNIX System Services pipes are:

- ▶ BatchPipes require an additional software licence and additional customization. UNIX System Services pipes are a standard feature of the UNIX System Services environment. BatchPipes are easy to use if you have already a BatchPipes environment active.
- ▶ UNLOAD to BatchPipes is not supported.





## Reorganizing data

The DB2 utility for reorganizing data is called the REORG utility. Its specific purpose is to improve access performance and to reclaim fragmented space. REORG TABLESPACE reorganizes data to improve the performance of a range scan that uses a clustering index to locate the rows in a table or a non-clustering index that is correlated with the clustering index. REORG INDEX reorganizes an index to improve the performance of an index when the index has become disorganized due to page splits and reclaim space by removing pseudo-deleted entries.

With the REORG utility, you can:

- ▶ Reorganize an entire table space
- ▶ Reorganize a partition or range of partitions of a partitioned table space
- ▶ Rebalance partitions
- ▶ Reorganize one or more indexes of a table space
- ▶ Reorganize a part of a partitioned index
- ▶ Reorganize catalog and directory spaces
- ▶ Delete rows using the DISCARD option
- ▶ Reorganize LOB and XML table spaces
- ▶ Compress a table space
- ▶ Convert a table space from BRF to RRF and vice versa
- ▶ Move objects to other DASD volumes

In this chapter, we mainly focus on new REORG options introduced in DB2 V8 and DB2 9. This chapter contains the following sections:

- ▶ Best practices when using the REORG utility
- ▶ The elimination of the BUILD2 phase in DB2 9
- ▶ Online REORG of LOB and XML data

## 8.1 Best practices when using the REORG utility

In this section, we discuss common best practices for availability and performance with REORG.

### 8.1.1 Use online REORG

In general, we suggest using REORG SHRLEVEL REFERENCE or REORG SHRLEVEL CHANGE because of the availability of the data during the REORG execution and after a REORG utility failure.

#### Availability of the base table space during the REORG

the base table space remains available for read-only applications (SHRLEVEL REFERENCE) or read-write applications (SHRLEVEL CHANGE) during most of the execution time of the REORG utility. There is just a small unavailability when the applications are drained in the final log iteration phase (CHANGE only) and the switch phase. By using drain and retry options, the draining process can be controlled without leading to resource unavailability conditions for the applications and the process can be repeated until the REORG finishes successfully.

#### Availability of the base table space after REORG failure

In most cases, the base table space remains available after the failure and the Online REORG can be terminated and all objects reverted to their original state. No additional recovery actions are necessary afterwards.

SHRLEVEL REFERENCE is ideal for objects that are read-only for long periods of time but have time available for the SWITCH phase to complete, such as data warehouse tables and tables that are loaded with LOAD RESUME YES.

SHRLEVEL CHANGE allows for continuous availability. It should be used in a situation where there is constant access to the DB2 object by updating transactions, and where a high availability is required by the users.

SHRLEVEL NONE is only recommended when the disk space for shadow data sets is not available or when you want to use the REUSE option to preserve the allocated VSAM data sets without deleting and defining them. Intervention is always required after a REORG failure before the data will be available again. This may involve restarting the utility or recovering of the objects.

### 8.1.2 How to get the best REORG performance

Different options should be taken into consideration when trying to optimize the performance of the REORG utility.

#### Invoke parallelism

In Chapter 4, “Performance via parallelism of executions” on page 75, we explain how parallelism is the major technique that DB2 uses to shorten the elapsed times of utilities. Maximizing exploitation of parallelism is of primary importance in reducing the elapsed time for the REORG utility.

Different forms of parallelism can be considered with the REORG utility:

- ▶ Inline COPY and inline RUNSTATS in parallel subtasks. Refer to 4.2.1, “Inline COPY” on page 77 and 4.2.2, “Inline RUNSTATS” on page 79 for more information.
- ▶ Parallel Index Build (PIB) to rebuild the different indexes of the table space in parallel. Refer to 4.6, “Parallelism with the REORG TABLESPACE utility” on page 85 for more information.
- ▶ Unload and reload the partitions of a partitioned table space in parallel. Refer to 4.6, “Parallelism with the REORG TABLESPACE utility” on page 85 for more information.

If you want to minimize the elapsed times of your REORG jobs, try to use parallelism as much as possible:

- ▶ Use REORG with LOG NO and COPYDDN to minimize the log volume and to create an inline copy so that the table space is fully available to all applications and fully recoverable. For online REORG (REORG SHRLEVEL REFERENCE and SHRLEVEL CHANGE), LOG NO and COPYDDN are mandatory.
- ▶ Use REORG with STATISTICS to keep the access path statistics current after REORG.
- ▶ Parallel Index Build will always be enabled.
- ▶ Parallel unload and load of the partitions of a partitioned table space will always be enabled for REORG SHRLEVEL CHANGE. For REORG SHRLEVEL REFERENCE or REORG SHRLEVEL NONE, you should use the NOSYSREC option or specify UNLDDN with a template name, where the template’s data set name pattern includes a partition number. If you specify NOSYSREC with REORG SHRLEVEL NONE, the job is not restartable, and you must take an image copy to ensure recoverability of your table space. You should always specify the SORTDEVT keyword. DB2 needs SORTDEVT to determine the optimal degree of parallelism and to dynamically allocate the sort work data sets by itself.
- ▶ Let DB2 allocate the optimal number of sort data sets during index build by eliminating the SORTNUM parameter and by removing sort workfiles from the JCL. Refer to Chapter 6, “Sort processing” on page 141 for more details and other considerations concerning SORT.
- ▶ Check if the degree of parallelism is not constrained by monitoring the DSNU3971 message in the job output of the REORG utility, as explained in “4.9, “Considerations for running parallel subtasks” on page 90.

In Example 8-1 we give an example of a REORG SHRLEVEL REFERENCE of a segmented table space. The PIB is always invoked by default. We specify NOSYSREC to pass the records to the RELOAD phase in memory and thus avoid I/O on a unload data set. We do not specify UNLDDN because the unload data set is not used. We also do not specify SORTDATA because this is the default and recommended. We use Inline Copy and Inline RUNSTATS as recommended. We do not specify SORTNUM and let DB2 allocate the optimal number of sort data sets. We have no workfiles allocated in the JCL.

*Example 8-1 Reorganization of a segmented table space*

---

```

TEMPLATE TSORTOUT DSN('&US..&SS..&DB..&SN..S&JU(3,5)..#&TI.')
      SPACE TRK MAXPRIME 65536
      DISP(MOD,DELETE,CATLG)
TEMPLATE TSYSUT1 DSN('&US..&SS..&DB..&SN..U&JU(3,5)..#&TI.')
      DISP(MOD,DELETE,CATLG)
TEMPLATE TSYSCOPY
      DSN('&US..&SS..&DB..&SN..&IC.&JU(3,5)..#&TI.')
      SPACE TRK MAXPRIME 65536
      DISP(MOD,CATLG,CATLG)
REORG  TABLESPACE MTSTBP.MTSTBP NOSYSREC SORTDEVT 3390
      COPYDDN(TSYSCOPY)
      SHRLEVEL REFERENCE
      DRAIN WAIT 20 RETRY 6 RETRY_DELAY 120 TIMEOUT TERM
      WORKDDN(TSYSUT1,TSORTOUT)
      STATISTICS TABLE(ALL) INDEX(ALL KEYCARD)

/*

```

---

An extract of the job's output can be seen in Example 8-2. We have no DSNU3097I message and the maximum degree of parallelism was used.

*Example 8-2 Job output of REORG with Parallel Index Build*

---

```

DSNU3340I 281 17:21:45.36 DSNUGSRT - UTILITY PERFORMS DYNAMIC ALLOCATION OF SORT DISK SPACE
DSNU252I 281 17:21:49.31 DSNUGSRT - UNLOAD PHASE STATISTICS - NUMBER OF RECORDS UNLOADED=1248796 FOR TABLESPACE
MTSTBP.MTSTBP
DSNU250I 281 17:21:49.31 DSNUGSRT - UNLOAD PHASE COMPLETE, ELAPSED TIME=00:00:03
DSNU3340I 281 17:21:49.44 DSNURPIB - UTILITY PERFORMS DYNAMIC ALLOCATION OF SORT DISK SPACE
DSNU3342I 281 17:21:49.87 DSNURPIB - NUMBER OF OPTIMAL SORT TASKS = 5, NUMBER OF ACTIVE SORT TASKS = 5
DSNU395I 281 17:21:49.87 DSNURPIB - INDEXES WILL BE BUILT IN PARALLEL, NUMBER OF TASKS = 15
DSNU400I 281 17:22:07.53 DSNURBID - COPY PROCESSED FOR TABLESPACE MTSTBP.MTSTBP
      NUMBER OF PAGES=67931
      AVERAGE PERCENT FREE SPACE PER PAGE = 4.80
      PERCENT OF CHANGED PAGES =100.00
      ELAPSED TIME=00:00:22
DSNU304I -DB9A 281 17:22:07.53 DSNURWT - (RE)LOAD PHASE STATISTICS - NUMBER OF RECORDS=1248796 FOR TABLE S.MTSTBP
DSNU302I 281 17:22:07.54 DSNURILD - (RE)LOAD PHASE STATISTICS - NUMBER OF INPUT RECORDS PROCESSED=1248796
DSNU300I 281 17:22:07.54 DSNURILD - (RE)LOAD PHASE COMPLETE, ELAPSED TIME=00:00:18
DSNU394I -DB9A 281 17:22:11.13 DSNURBXA - SORTBLD PHASE STATISTICS - NUMBER OF KEYS=1248796 FOR INDEX S.I_MTSTBP_5
DSNU394I -DB9A 281 17:22:11.73 DSNURBXA - SORTBLD PHASE STATISTICS - NUMBER OF KEYS=1248796 FOR INDEX S.I_MTSTBP_3
DSNU394I -DB9A 281 17:22:12.40 DSNURBXA - SORTBLD PHASE STATISTICS - NUMBER OF KEYS=1248796 FOR INDEX S.I_MTSTBP_4
DSNU394I -DB9A 281 17:22:13.00 DSNURBXA - SORTBLD PHASE STATISTICS - NUMBER OF KEYS=1248796 FOR INDEX S.I_MTSTBP_1
DSNU394I -DB9A 281 17:22:13.25 DSNURBXA - SORTBLD PHASE STATISTICS - NUMBER OF KEYS=1248796 FOR INDEX S.I_MTSTBP_2
DSNU391I 281 17:22:13.41 DSNURPTB - SORTBLD PHASE STATISTICS. NUMBER OF INDEXES = 5
DSNU392I 281 17:22:13.41 DSNURPTB - SORTBLD PHASE COMPLETE, ELAPSED TIME = 00:00:05
DSNU387I 281 17:22:13.68 DSNURSWT - SWITCH PHASE COMPLETE, ELAPSED TIME = 00:00:00
DSNU428I 281 17:22:13.68 DSNURSWT - DB2 IMAGE COPY SUCCESSFUL FOR TABLESPACE MTSTBP.MTSTBP

```

---

In Example 8-3, we give an example of a REORG SHRLEVEL CHANGE of a partitioned table space using parallel partition unloading and loading and Parallel Index Build (PIB). The parallel partition unloading and loading is invoked by default (NOSYSREC is forced) and so is the PIB. We also use Inline Copy and Inline RUNSTATS as recommended. We do not specify SORTNUM and let DB2 allocate the optimal number of sort data sets. We specify SORTDEVT. We have no workfiles allocated in the JCL.

*Example 8-3 Reorganization of a partitioned table space with full parallelism*

---

```

TEMPLATE TSORTOUT DSN('&US..&SS..&DB..&SN..S&JU(3,5)..#&TI.')
      SPACE TRK MAXPRIME 65536
      DISP(MOD,DELETE,CATLG)
TEMPLATE TSYSUT1 DSN('&US..&SS..&DB..&SN..U&JU(3,5)..#&TI.')
      DISP(MOD,DELETE,CATLG)
TEMPLATE TSYSCOPY
      DSN('&US..&SS..&DB..&SN..&IC.&JU(3,5)..#&TI.')
      SPACE TRK MAXPRIME 65536
      DISP(MOD,CATLG,CATLG)
REORG   TABLESPACE MTSTBO.MTSTBO
      COPYDDN(TSYSCOPY)
      SHRLEVEL CHANGE MAPPINGTABLE R.SIDDAGOY_SIDDAGO
      MAXRO 20 DRAIN ALL
      DRAIN_WAIT 20 RETRY 6 RETRY_DELAY 120 TIMEOUT TERM
      WORKDDN(TSYSUT1,TSORTOUT)
      SORTDEVT 3390
      STATISTICS TABLE(ALL) INDEX(ALL KEYCARD)

```

---

An extract of the job output is shown in Example 8-4.

*Example 8-4 Job output of REORG of a partitioned table space with full parallelism*

---

```

DSNU3340I  281 20:39:51.21 DSNURPCT - UTILITY PERFORMS DYNAMIC ALLOCATION OF SORT DISK SPACE
DSNU3342I  281 20:39:51.56 DSNURPCT - NUMBER OF OPTIMAL SORT TASKS = 31, NUMBER OF ACTIVE SORT TASKS = 4
DSNU1160I  281 20:39:51.58 DSNURPRD - PARTITIONS WILL BE UNLOADED/RELOADED IN PARALLEL, NUMBER OF TASKS= 2
DSNU395I   281 20:39:51.58 DSNURPRD - INDEXES WILL BE BUILT IN PARALLEL, NUMBER OF TASKS = 6
DSNU397I   281 20:39:51.58 DSNURPRD - NUMBER OF TASKS CONSTRAINED BY CPUS
DSNU251I  -DB9A 281 20:39:53.98 DSNURPUT - UNLOAD PHASE STATISTICS - NUMBER OF RECORDS UNLOADED=160767 FOR TABLESPACE
MTSTBO.MTSTBO PART 1
DSNU251I  -DB9A 281 20:39:53.98 DSNURPUT - UNLOAD PHASE STATISTICS - NUMBER OF RECORDS UNLOADED=224 FOR TABLESPACE
MTSTBO.MTSTBO PART 2
DSNU251I  -DB9A 281 20:39:53.98 DSNURPUT - UNLOAD PHASE STATISTICS - NUMBER OF RECORDS UNLOADED=2982 FOR TABLESPACE
MTSTBO.MTSTBO PART 3
.....;
.....
DSNU250I   281 20:39:53.99 DSNURPRD - UNLOAD PHASE COMPLETE, ELAPSED TIME=00:00:02
DSNU303I  -DB9A 281 20:40:11.19 DSNURWT - (RE)LOAD PHASE STATISTICS - NUMBER OF RECORDS=160767 FOR TABLE S.MTSTBO
PART=1
DSNU303I  -DB9A 281 20:40:11.19 DSNURWT - (RE)LOAD PHASE STATISTICS - NUMBER OF RECORDS=224 FOR TABLE S.MTSTBO PART=2
DSNU303I  -DB9A 281 20:40:11.19 DSNURWT - (RE)LOAD PHASE STATISTICS - NUMBER OF RECORDS=2982 FOR TABLE S.MTSTBO PART=3
DSNU303I  -DB9A 281 20:40:11.19 DSNURWT - (RE)LOAD PHASE STATISTICS - NUMBER OF RECORDS=18 FOR TABLE S.MTSTBO PART=4

```

---

In message DSNU397I, we see that parallelism is constrained by the number of available CPUs on the LPAR. The utility could start additional tasks, but it would not be efficient to do so, because we would overload the available CPUs.

A mapping table with an index must be created before using REORG SHRLEVEL CHANGE. DB2 uses this mapping table to map between the RIDs of data records in the original copy of the data and the corresponding RIDs in the shadow copy during the log apply phase. If you run multiple online REORG jobs in parallel, you need different mapping tables. We recommend creating a unique mapping table/index in the first step of your REORG job and drop it afterwards if the REORG succeeds successfully. If you use the sliding scale algorithm (subsystem parameter MSGEXTSZ=YES), you do not need to estimate PRIQTY and SECQTY parameters (refer to 7.4.3, "Using SLIDING SCALE" on page 181). One way of doing this task is to use the utility name (with a maximum of eight characters) or jobname of the REORG job as part of the mapping table name to provide the uniqueness of the mapping table. Put the mapping table space in a new database to avoid contention when executing the DDL. In Example 8-5, we show an example of how to do this task. Use EXEC SQL /ENDEXEC or DSNTIAD or their equivalents to execute the DDL in batch. Ensure that you have the proper privileges to create and drop the objects. Replace &UTILID by the actual utility name used or the jobname and use a value that does not yet exist.

---

*Example 8-5 Creating a unique mapping table for &UTILID*

---

```
CREATE DATABASE &UTILID ;
CREATE TABLESPACE MAPTBS IN &UTILID
    LOCKSIZE TABLE
    SEGSIZE 64 ;
CREATE TABLE MAPTB_&UTILID
    (TYPE          CHAR(1)          NOT NULL ,
     SOURCE_RID    CHAR(5)          NOT NULL ,
     TARGET_XRID   CHAR(9)          NOT NULL ,
     LRSN          CHAR(6)          NOT NULL )
    IN &UTILID.MAPTBS ;
CREATE UNIQUE INDEX MAPTBIX_&UTILID
    ON MAPTB_&UTILID
    (SOURCE_RID    ASC,
     TYPE          ASC,
     TARGET_XRID   ASC,
     LRSN          ASC) ;
--AFTER REORG STEP COMPLETES SUCCESSFULLY :
DROP DATABASE &UTILID ;
```

---

Tools like Automation Tool or DUET will use a similar technique to create a unique mapping table to be used by REORG SHRLEVEL CHANGE.



For REORG SHRLEVEL REFERENCE, we could use a REORG statement to achieve the unload/reload of the partitions, as shown in Example 8-6. We added NOSYSREC to trigger the parallelism.

*Example 8-6 REORG SHRLEVEL REFERENCE of a partitioned table space with NOSYSREC*

---

```

TEMPLATE TSORTOUT DSN('&US..&SS..&DB..&SN..S&JU(3,5)..#&TI.')
```

```

    SPACE TRK MAXPRIME 65536
    DISP(MOD,DELETE,CATLG)
TEMPLATE TSYSUT1  DSN('&US..&SS..&DB..&SN..U&JU(3,5)..#&TI.')
```

```

    DISP(MOD,DELETE,CATLG)
TEMPLATE TSYSCOPY
    DSN('&US..&SS..&DB..&SN..&IC.&JU(3,5)..#&TI.')
```

```

    SPACE TRK MAXPRIME 65536
    DISP(MOD,CATLG,CATLG)
REORG    TABLESPACE MTSTBO.MTSTBO NOSYSREC
        COPYDDN(TSYSCOPY)
        SHRLEVEL REFERENCE
        DRAIN_WAIT 20 RETRY 6 RETRY_DELAY 120 TIMEOUT TERM
        WORKDDN(TSYSUT1,TSORTOUT)
        SORTDEVT 3390
        STATISTICS TABLE(ALL) INDEX(ALL KEYCARD)
```

---

If for some reason you decide not to use NOSYSREC on the REORG SHRLEVEL REFERENCE and you want to achieve parallel unload and reload of the partitions, you should use a template for the unload data sets UNLDDN, as shown in Example 8-7. The TSYSREC template's data set name pattern includes the partition number &PART. to use a separate unload data set per parallel unload task.

*Example 8-7 REORG SHRLEVEL REFERENCE of a partitioned table space without NOSYSREC*

---

```

//SYSIN DD *
TEMPLATE TSYSREC DSN('&US..&SS..&DB..&SN..R&JU(3,5)..#&PART.')
```

```

    DISP(MOD,DELETE,CATLG)
TEMPLATE TSORTOUT DSN('&US..&SS..&DB..&SN..S&JU(3,5)..#&TI.')
```

```

    SPACE TRK MAXPRIME 65536
    DISP(MOD,DELETE,CATLG)
TEMPLATE TSYSUT1  DSN('&US..&SS..&DB..&SN..U&JU(3,5)..#&TI.')
```

```

    DISP(MOD,DELETE,CATLG)
TEMPLATE TSYSCOPY
    DSN('&US..&SS..&DB..&SN..&IC.&JU(3,5)..#&TI.')
```

```

    SPACE TRK MAXPRIME 65536
    DISP(MOD,CATLG,CATLG)
REORG    TABLESPACE MTSTBO.MTSTBO
        COPYDDN(TSYSCOPY)
        SHRLEVEL REFERENCE
        DRAIN_WAIT 20 RETRY 6 RETRY_DELAY 120 TIMEOUT TERM
        WORKDDN(TSYSUT1,TSORTOUT) UNLDDN(TSYSREC)
        SORTDEVT 3390
        STATISTICS TABLE(ALL) INDEX(ALL KEYCARD)
```

---

## Other things that influence the performance

Other actions that will influence the performance of the REORG utility are:

- ▶ Use the REUSE option for REORG SHRLEVEL NONE. This is similar to the LOAD utility. Refer to 7.1.2, “Use the REUSE option with LOAD” on page 174. This option cannot be used with REORG SHRLEVEL REFERENCE or CHANGE because of the shadow data sets.
- ▶ Use the NOSYSREC option, which is optional for REORG SHRLEVEL REFERENCE and SHRLEVEL NONE. This option will pass the keys in memory to the RELOAD phase, thereby avoiding I/O on the unload data set. When using the NOSYSREC option, there is no need to specify the UNLDDN unload data set. For a table space without a clustering index, the NOSYSREC option is ignored and you must specify the UNLDDN unload data set.
- ▶ USE SORTDATA YES, which is the default. This option will unload the table space with a table space scan and sort it afterwards in clustering sequence. SORT DATA NO can only be used with REORG SHRLEVEL REFERENCE or REORG SHRLEVEL NONE and is only recommended when the data is in near clustering order and there is an insufficient amount of sort disk space available.
- ▶ Enable Parallel Access Volume (PAV) on your DASD devices.
- ▶ If the data is compressed, specify KEEPDICTIONARY to avoid rebuilding the compression dictionary.
- ▶ Be sure to have the following maintenance applied: PK47083, PK60956, PK61759, PK67154, PK70866, PK79136, and PK85889.

### 8.1.3 Controlling online REORG

By using drain and retry options, the draining process can be controlled without leading to resource unavailability conditions for the applications and the process can be repeated until the REORG finishes successfully. We recommend carefully using the following options for REORG SHRLEVEL CHANGE:

- ▶ DRAIN\_WAIT: Specifies the number of seconds that the utility waits when trying to drain the table space or index at the last iteration of the log processing (and switch phase). Set this parameter 5-10 seconds less than the application timeout parameter IRLMWRT to avoid having applications time out by the drain operation. When this waiting time is over and the drain does not succeed, DB2 will give up the drain and return to processing the log. The drain is then retried after the time specified in the RETRY DELAY interval.
- ▶ MAXRO: During log processing, DB2 calculates the amount of time that will be taken to apply the log records not yet applied. If this value is less than the MAXRO threshold, the last log iteration will be started and DB2 will drain the applications according to the DRAIN option. Set this parameter 5-10 seconds less than the application timeout parameter IRLMWRT to avoid having applications time out by the drain operation during the last log iteration.
- ▶ DRAIN: Specifies drain behavior during the last log iteration. We recommend using DRAIN ALL, which means DB2 will try to drain all readers and writers during the last log iteration. Using DRAIN ALL increases the chances that the drain will be successful, as opposed to DRAIN WRITERS.
- ▶ RETRY: Specifies the maximum number of retries that REORG is to attempt before it terminates the REORG utility in accordance with the TIMEOUT parameter. We recommend using the default value, which is identical to the UTIMOUT subsystem parameter (utility lock timeout multiplier, which is 6 by default). Specifying too high a value can lead to increased processing costs and can result in multiple or extended periods of waiting times for the applications during the last log iteration.

- ▶ **RETRY\_DELAY:** Specifies the minimum duration, in seconds, between retries. We recommend using the **DRAIN\_WAIT** value in seconds multiplied by the **RETRY** value.
- ▶ **TIMEOUT:** Specifies the action to be taken if the **REORG** does not succeed in draining the table space or index at the last retry interval. We recommend using **TERM**, which means that DB2 will terminate the utility, including the cleanup of the shadow data sets, and leave the object in **RW** status. This action is in opposition to **TIMEOUT ABEND**, which will leave the object in **UTRO** or **UTUT** status and lead to resource unavailable conditions for the active applications.

For **REORG SHRLEVEL REFERENCE**, the same options, except **MAXRO** and **DRAIN**, can be used to control the drain behavior in the beginning of the utility, where all writers are drained, and at the switch phase, where the readers are drained as well.

See Example 8-1 on page 208 to Example 8-7 on page 211 for practical use of these options. The application timeout parameter **IRLMWRT** is set to 25 seconds.

With DB2 9, the Online **REORG** will always use drain and retry options, even when not specified. The defaults are:

- ▶ **DRAIN\_WAIT:** The application timeout parameter **IRLMWRT**
- ▶ **MAXRO:** The **RETRY\_DELAY** value
- ▶ **DRAIN:** **WRITERS**
- ▶ **RETRY:** The utility lock timeout multiplier **UTIMOUT**
- ▶ **RETRY\_DELAY:** The smallest of  $\text{DRAIN\_WAIT} \times \text{RETRY}$  and  $\text{DRAIN\_WAIT} \times 10$
- ▶ **TIMEOUT:** **TERM**

The chances of Online **REORGs** completing successfully is greatly increased if there are good application standards regarding commit frequency and the closing of **CURSORS** with **HOLD**. Do not forget that even applications that use the uncommit read (**UR**) isolation level are holding claims that can prevent the drains from being successful. Long units of work will inhibit the drains to be obtained. Therefore, we recommend that Online **REORGs** should not be run while long-running units of work are executing.

To detect long units of work, the DB2 subsystem should have the **DSNZPARM UR** checkpoint count option enabled. When enabled, DB2 generates console message **DSNR035I** and trace records for **IFCID 0313** to inform you about long-running **URs**. The **UR** checkpoint count option is enabled through subsystem parameter **URCHKTH**, where you can specify the number of checkpoint cycles that are to complete before DB2 issues the warning message to the console and instrumentation for an uncommitted unit of recovery (**UR**). The **IFCID 0313** (which is included in Statistics class(3)) will report readers that may block commands and utilities from draining, including **WITH HOLD** cursors into which a drain cannot break.

Starting with DB2 V8, DB2 will automatically issue a **DISPLAY CLAIMERS** command when the **REORG** job fails to drain an object and puts the output of the command in the utility job output. This will also help you in finding the applications that might need to be revised.

You could also consider postponing the **SWITCH** phase to a maintenance window to avoid concurrent workloads that may prevent the utility from breaking in:

- ▶ Specify that **MAXRO DEFER LONGLOG CONTINUE** and **REORG** will continue with applying log records; it will never start the last log iteration and the switch phase.
- ▶ Change the **MAXRO** value with the “**ALTER UTILITY MAXRO integer**” command to trigger the last log iteration phase.

Many log iterations might reduce the “perfect” organization of the table space, so keep the time until MAXRO is changed to allow final processing down to a minimum.

## 8.1.4 Other recommendations

Here we discuss other recommendations.

### REORG on the PART level

With DB2 9, all NPIs are completely recreated for online REORG jobs on the partition level (refer to 8.2, “The elimination of the BUILD2 phase in DB2 9” on page 215), even when only one partition is reorganized. Therefore, we recommend that in DB2 9 to reorganize as many contiguous partitions as possible in the same online REORG job, or even do an online REORG of the complete table space instead of using different jobs on different partitions or partition ranges. Execution of online REORG PART in parallel jobs on different partitions of the same table space is no longer allowed because of the NPI shadow data sets. REORG INDEX of the NPIs after a REORG PART is no longer necessary.

You can REORG in one REORG statement, with up to 4096 possible compressed partitions.

### REORG of LOB table spaces

Starting with DB2 9, we recommend using the new REORG SHRLEVEL REFERENCE when reorganizing LOB table spaces. Refer to 8.3, “Online REORG of LOB and XML data” on page 217 for more information.

### REORG after adding columns to a table

Run MODIFY RECOVERY some time after ALTER TABLE ADD COLUMN, which will improve REORG performance after adding a column. After you add a column to a table space, the next REORG of the table space creates default values for the added column by converting all of the fields in each row to the external DB2 format during the UNLOAD phase and then converting them to the internal DB2 format during the RELOAD phase. Subsequently, each REORG job for the table space repeats this processing in the UNLOAD and RELOAD phases, until all image copies prior to the ALTER TABLE are deleted from SYSCOPY and a point-in-time RECOVERY to a logpoint before the ALTER table is no longer possible.

### REORG of the catalog and directory

Catalog and directory table spaces need only be reorganized infrequently. The reasons may be:

- ▶ A REORG of DSNDB01.SYSLGRNGX is needed to speed up the MODIFY utility.
- ▶ To reduce the size of the catalog and to speed up the CATMAINT processing when migrating to a new version of DB2.
- ▶ For reclaiming space or for the performance of queries that use table space scans against the catalog.

Catalog and directory indexes should be reorganized for the same reasons as applications' indexes. When catalog indexes become disorganized, this affects both the performance of queries against the catalog and DB2 performance when index access is used.

There are restrictions and special considerations to be taken into account when reorganizing the DB2 catalog and directory. Refer to *DB2 Version 9.1 for z/OS Utility Guide and Reference*, SC18-9855 for more information.

## Using REORG to convert from BRF to RRF and vice versa

As explained in 7.2, “Loading into reordered row format” on page 176, REORG TABLESPACE can also be used to convert a table space from basic row format to reordered row format and vice versa.

## 8.2 The elimination of the BUILD2 phase in DB2 9

Prior to DB2 9, when you reorganized a table space with REORG TABLESPACE PART n SHRLEVEL REFERENCE or CHANGE, serious outages were possible on the NPIs. This because of the BUILD2 phase, which was only invoked when NPIs were present and when the table space was reorganized on the partition level.

When there were NPIs, shadow data sets were created for the NPIs in which index entries were created for the logical partitions corresponding to the partitions being reorganized. These shadow NPIs only contained a subset of the NPI. The function of the BUILD2 phase was to update the original NPIs from the NPI shadow data sets with the new record identifiers (RIDs) that are generated by REORG. During BUILD2, the entire NPI was unavailable for write operations and this outage was in addition to the outage during the SWITCH phase, as shown in Figure 8-1. For SHRLEVEL REFERENCE, there was no LOG phase, because the involved table space partitions and equivalent logical partitions in the NPIs were unavailable for write operations from the beginning of the utility and updates done on the other partitions were done directly in the base objects and not in shadow objects.

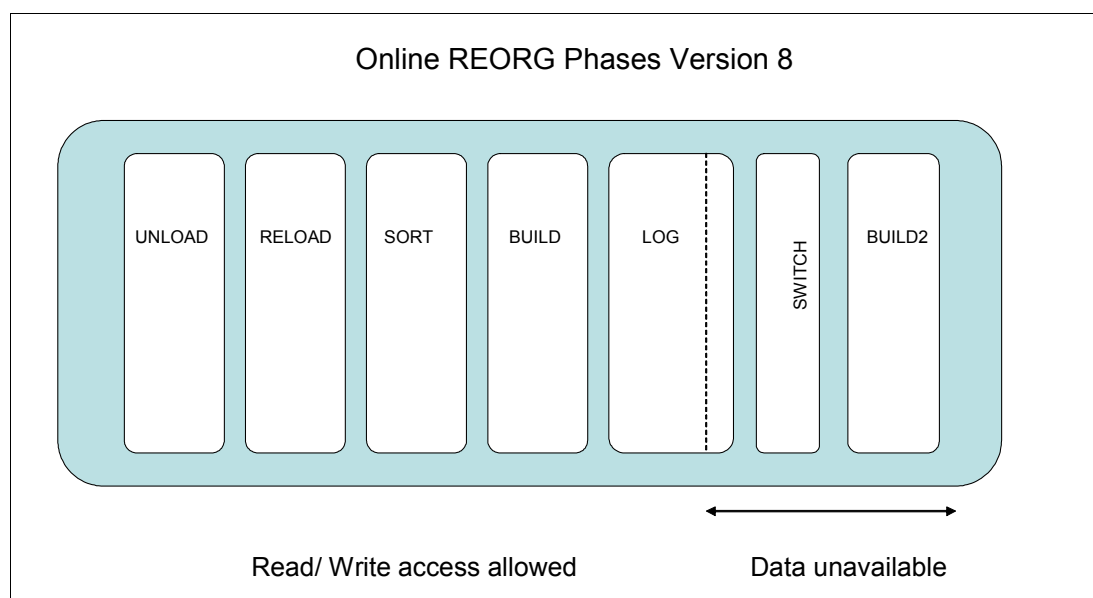


Figure 8-1 REORG TABLESPACE PART n SHRLEVEL CHANGE phases in DB2 V8

DB2 V8 introduced data-partitioned secondary indexes that can be used to avoid the presence of NPIs, but using them often involves application design changes, because query-wise, a data-partitioned secondary index is most useful when the query has predicates on both the secondary index column(s) and the partitioning index column(s).

In DB2 9, actions were taken to eliminate the BUILD2 phase completely for REORG SHRLEVEL REFERENCE or CHANGE by creating shadow data sets for the NPIs, which contain a complete copy of the NPIs, instead of the subsets, as shown in Figure 8-2. These NPI shadow data sets are created during the unload phase, updated during the log apply phase, and switched during the switch phase. Updating of the NPIs during the log apply phase happens through parallel subtasks. Even with SHRLEVEL REFERENCE, there is always a LOG phase to reflect the index updates done on the other partitions in the shadow NPIs. But a mapping table is not necessary because the RIDs will not change. All this also means that the cost of online REORG PART n utilities will go up in DB2 9, and that additional disk space for the NPI shadow data sets is necessary.

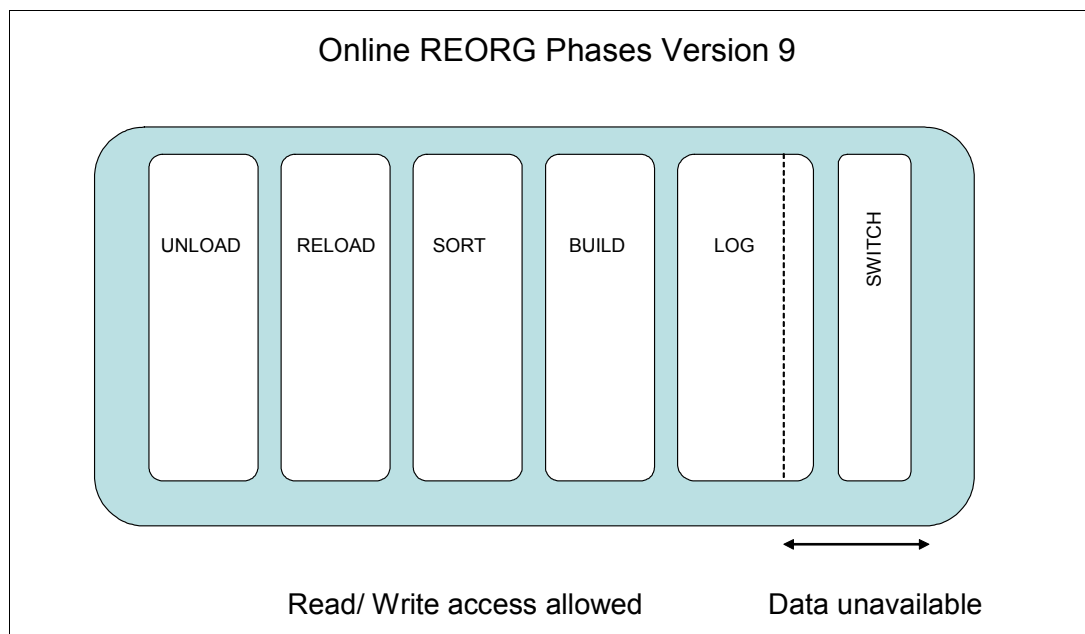


Figure 8-2 REORG TABLESPACE PART n phases in DB2 9

The unload/reload of the whole NPI during a REORG SHRLEVEL CHANGE (or REFERENCE) PART is essentially equivalent to a REORG INDEX of the NPI. If you currently run REORG INDEX on all NPIs following a REORG PART, this action should no longer be needed, and we recommend changing your existing jobs.

Because all NPIs are recreated during a REORG TABLESPACE PART n, we recommend, in DB2 9, reorganizing as many contiguous partitions as possible in the same Online REORG job, or even do a Online REORG of the complete table space instead of using different jobs on different partitions or partition ranges.

Execution of online REORG PART in parallel jobs on different partitions of the same table space is no longer allowed because of the unique NPI shadow data sets. If you do that task, only the first job will execute, and the others will fail with RC=8 and message DSNU180I - UTILITY IS NOT COMPATIBLE WITH THE REORG TABLESPACE UTILITY ,UTILID = ..., OBJECT = npi name.

## 8.3 Online REORG of LOB and XML data

With DB2 9, utilities have been largely improved for handling LOBs and have also included the functionalities for dealing with the XML data type.

### 8.3.1 Online REORG of a LOB table space

With DB2 9, it is now possible to do a REORG SHRLEVEL REFERENCE of a LOB table space. The REORG is similar to the REORG of regular table spaces, meaning that LOB data is unloaded and reloaded into a shadow data set, which is switched at the end to become the base object. If you specify SHRLEVEL REFERENCE, a REORG of a LOB table space will make LOB pages continuous, remove imbedded free space, and reclaim physical space if applicable. In Figure 8-3, we illustrate the LOB architecture where the actual LOB data is stored in a separate LOB table space linked with the base table by the auxiliary index. the XML documents will be stored in a separate implicitly created XML table space and the XML documents are linked via the document ID index.

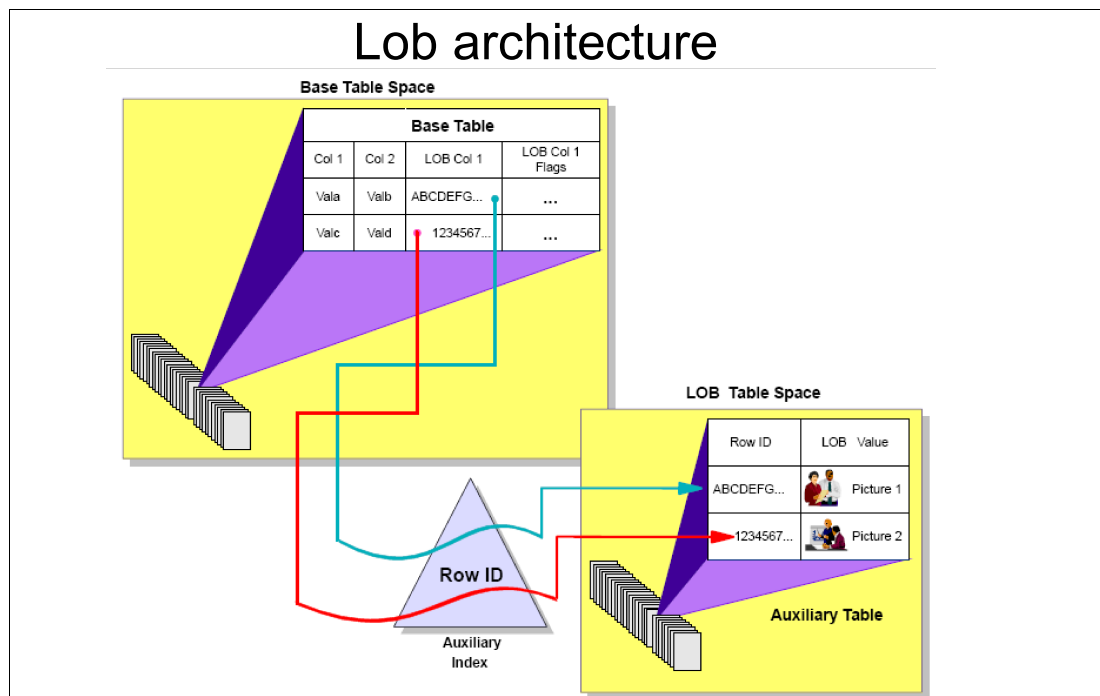


Figure 8-3 LOB architecture

As with regular table spaces and indexes, the REORG SHRLEVEL REFERENCE utility can be used to:

- ▶ Reorganize a LOB table space to reclaim fragmented space and improve access performance
- ▶ Reorganize an auxiliary index to reclaim fragmented space and improve access performance

REORG of LOB table spaces was introduced in DB2 V6 and it was quite different from the REORG of regular table spaces. Although for migration reasons this REORG method is still supported in DB2 9 and known as REORG SHRLEVEL NONE, our recommendation is to no longer use this REORG method and start using REORG SHRLEVEL REFERENCE in DB2 9. REORG SHRLEVEL NONE's main characteristics were:

- ▶ No access to LOB data was allowed while the REORG executed.
- ▶ It was an inline REORG, which means that LOBS were moved within the existing LOB table space without unload and reload and without delete and define of the underlying VSAM cluster; there was no means of reclaiming physical space from the LOB data set by resizing it. ALTER of PRIQTY and SECQTY values of the LOB table space were not taken into account. Its aim was to store all pages belonging to an individual LOB as contiguous pages (chunking) to improve the prefetch performance of the LOB data; this resulted in a tradeoff between optimal reorganization and physical space consumption, because it did not always remove all holes between LOBs.
- ▶ It was always LOG YES, resulting in significant additional logging, particularly for LOB table spaces defined as LOG YES (V7 and V8) or LOGGED (DB2 9). Inline image copy using the COPYDDN keyword was not allowed.
- ▶ The utility was not restartable during the REORGLOB phase and the LOB was left in RECP status if a failure happened; a recovery was always needed afterwards in case of a failure.

Chunking is illustrated in Figure 8-4.

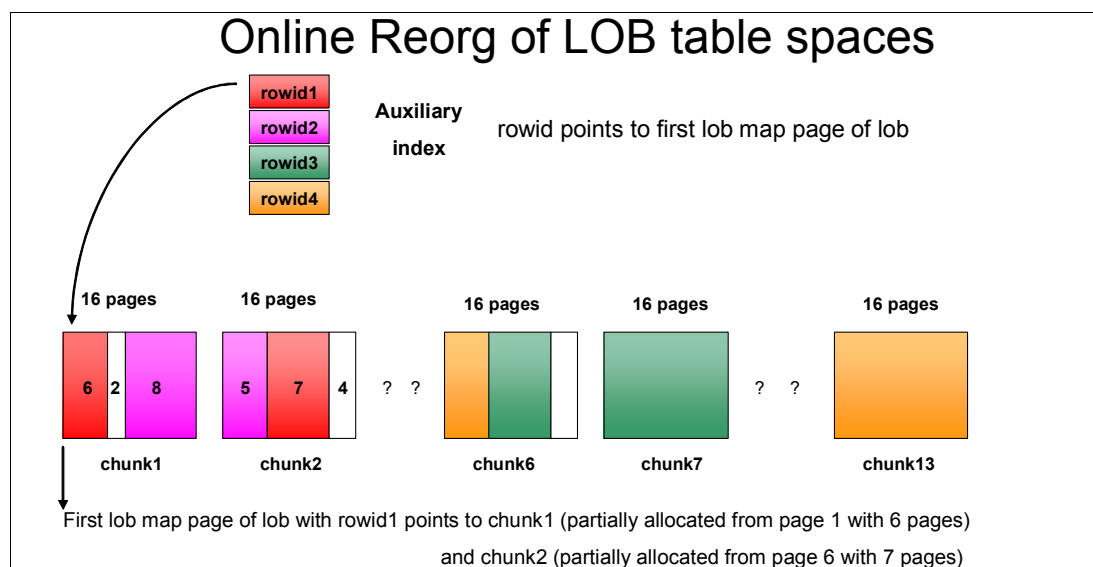


Figure 8-4 Chunks of LOB pages

DB2 allocates space for LOB data in chunks. A chunk is 16 contiguous pages (16 immediately adjacent pages). A single LOB value can be stored in multiple chunks, which will be fully or partially allocated to that LOB. When an application wants to access a certain LOB, the ROWID stored in the base table will be used to locate the first LOB map page via the auxiliary index. This LOB map page contains a table with all the chunks that are fully or partially allocated to this LOB. A LOB table space is considered to be perfectly organized when the number of chunks allocated to a lob is minimal (no extra chunks needed). This is expressed by the ORGRATIO factor, which is 100 in the perfect case. In Figure 8-4, the LOB pointed by rowid1 is not perfectly organized because it is allocated to two chunks (six pages in chunk1 and seven pages in chunk2) where one chunk (13 out of 16 pages) would be sufficient.



ORGRATIO is the percentage of organization in the LOB table space. A value of 100 indicates perfect organization of the LOB table space. A value of 1 indicates that the LOB table space is disorganized. A value of 0.00 indicates that the LOB table space is totally disorganized.

With the new REORG SHRLEVEL REFERENCE method, the original LOB table space is drained of writers, that is, no update access is allowed during the REORG. All LOBs are then extracted from the original data set and inserted into a shadow data set. A new auxiliary index is also built in a shadow data set. Once this is complete, all access to the LOB table space is stopped for a short period while the original data sets are switched with the shadows. At this point, full access to the new data set is allowed. An inline copy is taken to ensure recoverability. The allocation of the shadow and deallocation of the old original data set follow the same rules that exist today for REORG SHRLEVEL REFERENCE and CHANGE of regular table spaces.

The new method has the following benefits:

- ▶ REORG SHRLEVEL REFERENCE allows full read access to LOB data for the duration of the REORG, with the exception of the switch phase, during which readers are also drained. Once the switch has occurred, full access is restored.
- ▶ The LOBs are copied from the original to the shadow table space. This results in an implicit reorganization. There is no sorting as with regular table spaces. The existing LOB allocation algorithms are used.
- ▶ Physical space is reclaimed, because after REORG, the original data set is deleted and the shadow data set is governed by normal space allocation rules. Shadow data sets are allocated according to normal space allocation rules for the LOB table space. Changes made to PRIQTY and SECQTY with ALTER before the REORG are honored. RECOVER is therefore no longer needed to resize a LOB table space.
- ▶ As with the pre-DB2 9 REORG of LOB table spaces, the new solution has no dependency on the base table or base table indexes. Therefore, REORG of a LOB table space continues to have no impact on access to base table data.
- ▶ LOG YES is not valid for REORG SHRLEVEL REFERENCE of a LOB table space. This results in reduced logging requirements for LOB table spaces with no loss of recoverability.
- ▶ An inline copy is always taken to ensure recoverability. This is required and the COPYDDN parameter is needed unless a SYSCOPY DD card is specified in the utility JCL. As with a normal REORG SHRLEVEL REFERENCE, the REORG of a LOB table space now inserts two SYSCOPY records. The first SYSCOPY record represents the REORG itself, which is a nonrecoverable event, and the second SYSCOPY record represents the inline image copy.
- ▶ The utility cannot be restarted when it fails. The shadow pageset is discarded in event of failure.

For examples and other considerations about reorganizing LOB data, refer to *LOBs with DB2 for z/OS: Stronger and Faster*, SG24-72700.

### 8.3.2 Online REORG of an XML table space

If one creates a table that contains one or more columns of the new DB2 9 XML data type, for each XML column, an XML table space is implicitly created to store the XML data. Furthermore, an XML table, a node ID index, and document ID index are implicitly created as well.

For XML table spaces, SHRLEVEL NONE, SHRLEVEL REFERENCE, and SHRLEVEL CHANGE can be used. Do not forget to:

- Add a COPYDDN for producing an inline copy during REORG SHRLEVEL REFERENCE or CHANGE
- Use a mapping table for REORG SHRLEVEL CHANGE

The DDL in Example 8-8 shows that table space DSN8D91X.XPRO0000, with an implicitly created XML table space, belongs to the column DESCRIPTION of sample table DSN8910.PRODUCT.

*Example 8-8 DDL of table DSN8910.PRODUCT containing a XML column*

---

```
CREATE TABLE DSN8910.PRODUCT
(PID                VARCHAR(10) FOR SBCS DATA NOT NULL,
NAME               VARCHAR(128) FOR SBCS DATA WITH DEFAULT NULL,
PRICE             DECIMAL(30, 2) WITH DEFAULT NULL,
PROMOPRICE        DECIMAL(30, 2) WITH DEFAULT NULL,
PROMOSTART        DATE WITH DEFAULT NULL,
PROMOEND          DATE WITH DEFAULT NULL,
DESCRIPTION        XML,
CONSTRAINT PID
PRIMARY KEY (PID))
IN DSN8D91X.DSN8S91X ;
CREATE INDEX DSN8910.PROD_NAME_XMLIDX
ON DSN8910.PRODUCT(DESCRIPTION)
GENERATE KEY USING XMLPATTERN '/product/description/name'
AS SQL VARCHAR(128)
USING STOGROUP DSN8G910;
CREATE INDEX DSN8910.PROD_DETAIL_XMLIDX
ON DSN8910.PRODUCT(DESCRIPTION)
GENERATE KEY USING XMLPATTERN '/product/description/detail'
AS SQL VARCHAR(128)
USING STOGROUP DSN8G910;
```

---

In Example 8-9, we show the job output for a REORG SHRLEVEL CHANGE of table space DSN8D91X.XPRO0000.

*Example 8-9 Job output for REORG SHRLEVEL CHANGE of an XML table space*

---

```
DSNU050I   285 17:13:02.07 DSNUGUTC - REORG TABLESPACE DSN8D91X.XPRO0000 NOSYSREC COPYDDN(TSYS COPY)
SHRLEVEL CHANGE MAPPINGTABLE R.SIDDAGOY_SIDDAGO
MAXRO 20 DRAIN ALL DRAIN_WAIT 20 RETRY 120 RETRY_DELAY 60 TIMEOUT TERM WORKDDN(TSYSUT1, TSORTOUT) SORTDEVT
3390 STATISTICS TABLE(ALL) INDEX(ALL)
DSNU1038I  285 17:13:02.69 DSNUGDYN - DATASET ALLOCATED. TEMPLATE=TSYSCOPY
          DDNAME=SYS00001
          DSN=DB2R6.DB9A.DSN8D91X.XPRO0000.F09285.#211326
DSNU3340I  285 17:13:02.70 DSNUGSRT - UTILITY PERFORMS DYNAMIC ALLOCATION OF SORT DISK SPACE
DSNU252I   285 17:13:02.78 DSNUGSRT - UNLOAD PHASE STATISTICS - NUMBER OF RECORDS UNLOADED=6 FOR
TABLESPACE DSN8D91X.XPRO0000
DSNU250I   285 17:13:02.78 DSNUGSRT - UNLOAD PHASE COMPLETE, ELAPSED TIME=00:00:00
DSNU3340I  285 17:13:02.84 DSNURPIB - UTILITY PERFORMS DYNAMIC ALLOCATION OF SORT DISK SPACE
DSNU3342I  285 17:13:03.01 DSNURPIB - NUMBER OF OPTIMAL SORT TASKS = 4, NUMBER OF ACTIVE SORT TASKS = 4
DSNU395I   285 17:13:03.01 DSNURPIB - INDEXES WILL BE BUILT IN PARALLEL, NUMBER OF TASKS = 12
DSNU304I   -DB9A 285 17:13:03.09 DSNURWT - (RE)LOAD PHASE STATISTICS - NUMBER OF RECORDS=6 FOR TABLE
DSN8910.XPRODUCT
DSNU302I   285 17:13:03.09 DSNURILD - (RE)LOAD PHASE STATISTICS - NUMBER OF INPUT RECORDS PROCESSED=6
DSNU300I   285 17:13:03.09 DSNURILD - (RE)LOAD PHASE COMPLETE, ELAPSED TIME=00:00:00
DSNU394I   -DB9A 285 17:13:03.18 DSNURBXA - SORTBLD PHASE STATISTICS - NUMBER OF KEYS=6 FOR INDEX
```

```

DSN8910.I_NODEIDXPRODUCT
DSNU394I -DB9A 285 17:13:03.20 DSNURBXE - SORTBLD PHASE STATISTICS - NUMBER OF KEYS=0 FOR INDEX
DSN8910.PROD_NAME_XMLIDX
DSNU394I -DB9A 285 17:13:03.22 DSNURBXA - SORTBLD PHASE STATISTICS - NUMBER OF KEYS=6 FOR INDEX
R.ISIDDAGOY_SIDDAGO
DSNU394I -DB9A 285 17:13:03.22 DSNURBXE - SORTBLD PHASE STATISTICS - NUMBER OF KEYS=0 FOR INDEX
DSN8910.PROD_DETAIL_XMLIDX
DSNU391I 285 17:13:03.27 DSNURPTB - SORTBLD PHASE STATISTICS. NUMBER OF INDEXES = 4
DSNU392I 285 17:13:03.27 DSNURPTB - SORTBLD PHASE COMPLETE, ELAPSED TIME = 00:00:00
DSNU386I 285 17:13:03.38 DSNURLGD - LOG PHASE STATISTICS. NUMBER OF ITERATIONS = 1, NUMBER OF LOG
RECORDS = 0
DSNU385I 285 17:13:03.38 DSNURLGD - LOG PHASE COMPLETE, ELAPSED TIME = 00:00:00
DSNU400I 285 17:13:03.40 DSNURBID - COPY PROCESSED FOR TABLESPACE DSN8D91X.XPRO0000
        NUMBER OF PAGES=9
        AVERAGE PERCENT FREE SPACE PER PAGE = 9.88
        PERCENT OF CHANGED PAGES =100.00
        ELAPSED TIME=00:00:00
DSNU387I 285 17:13:03.49 DSNURSWT - SWITCH PHASE COMPLETE, ELAPSED TIME = 00:00:00
DSNU428I 285 17:13:03.49 DSNURSWT - DB2 IMAGE COPY SUCCESSFUL FOR TABLESPACE DSN8D91X.XPRO0000
DSNU610I -DB9A 285 17:13:03.53 DSNUSUTP - SYSTABLEPART CATALOG UPDATE FOR DSN8D91X.XPRO0000 SUCCESSFUL
DSNU610I -DB9A 285 17:13:03.53 DSNUSUPT - SYSTABSTATS CATALOG UPDATE FOR DSN8910.XPRODUCT SUCCESSFUL
DSNU610I -DB9A 285 17:13:03.53 DSNUSUPC - SYSCOLSTATS CATALOG UPDATE FOR DSN8910.XPRODUCT SUCCESSFUL
DSNU610I -DB9A 285 17:13:03.53 DSNUSUTB - SYSTABLES CATALOG UPDATE FOR DSN8910.XPRODUCT SUCCESSFUL
DSNU610I -DB9A 285 17:13:03.54 DSNUSUCO - SYSCOLUMNS CATALOG UPDATE FOR DSN8910.XPRODUCT SUCCESSFUL
DSNU610I -DB9A 285 17:13:03.54 DSNUSUTS - SYSTABLESPACE CATALOG UPDATE FOR DSN8D91X.XPRO0000 SUCCESSFUL
DSNU610I -DB9A 285 17:13:03.55 DSNUSUIP - SYSINDEXPART CATALOG UPDATE FOR DSN8910.I_NODEIDXPRODUCT
SUCCESSFUL
DSNU610I -DB9A 285 17:13:03.55 DSNUSUIP - SYSINDEXPART CATALOG UPDATE FOR DSN8910.PROD_NAME_XMLIDX
SUCCESSFUL
DSNU610I -DB9A 285 17:13:03.56 DSNUSUIP - SYSINDEXPART CATALOG UPDATE FOR DSN8910.PROD_DETAIL_XMLIDX
SUCCESSFUL
DSNU610I -DB9A 285 17:13:03.59 DSNUSUKT - SYSKEYTARGETS CATALOG UPDATE FOR DSN8910.I_NODEIDXPRODUCT
SUCCESSFUL
DSNU610I -DB9A 285 17:13:03.59 DSNUSUIX - SYSINDEXES CATALOG UPDATE FOR DSN8910.I_NODEIDXPRODUCT
SUCCESSFUL
DSNU610I -DB9A 285 17:13:03.60 DSNUSUKT - SYSKEYTARGETS CATALOG UPDATE FOR DSN8910.PROD_NAME_XMLIDX
SUCCESSFUL
DSNU610I -DB9A 285 17:13:03.60 DSNUSUIX - SYSINDEXES CATALOG UPDATE FOR DSN8910.PROD_NAME_XMLIDX
SUCCESSFUL
DSNU610I -DB9A 285 17:13:03.60 DSNUSUKT - SYSKEYTARGETS CATALOG UPDATE FOR DSN8910.PROD_DETAIL_XMLIDX
SUCCESSFUL
DSNU610I -DB9A 285 17:13:03.60 DSNUSUIX - SYSINDEXES CATALOG UPDATE FOR DSN8910.PROD_DETAIL_XMLIDX
SUCCESSFUL
DSNU620I -DB9A 285 17:13:03.60 DSNUSEOF - RUNSTATS CATALOG TIMESTAMP = 2009-10-12-17.13.02.840462
DSNU010I 285 17:13:04.11 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0

```

---

In the above example, we do not show the TEMPLATE definitions.





## Copying data

In this chapter, we show and discuss the options available when copying DB2 data.

We look first at the COPY utility and the methods used by the COPY utility. We then discuss the COPYTOCOPY utility, which allows you to make asynchronous extra image copies, and we briefly examine the Concurrent COPY option, and finally some DSN1COPY enhancements.

This chapter contains the following sections:

- ▶ COPY
- ▶ COPYTOCOPY
- ▶ Concurrent COPY
- ▶ DSN1COPY

## 9.1 COPY

The COPY utility is used to take secure images of table spaces and index spaces to allow for the recovery of data in the event of data loss or corruption. With DB2 9, the COPY utility uses an MRU algorithm for the management of the buffer pool usage, therefore limiting the impact on concurrently running online applications.

### 9.1.1 Inline COPY with LOAD and REORG

This is the ability to take image copies during the LOAD (REPLACE or RESUME NO) utility and during the REORG utility. This function reduces the time the object is unavailable to the applications due to the COPY pending flag being set.

Previously, after the execution of one of these utilities, assuming LOG NO is specified for speed, an image copy had to be taken to ensure the integrity of the data. The time taken to run the copy added to the outage for the object. The Inline COPY solved this problem.

When a LOAD REPLACE/LOAD RESUME NO is executed with an Inline COPY, the copy is taken during the LOAD phase. This means that if there are any rows on the table that are later discarded, these will still remain on the image copy. For this reason, we do *not* recommend using inline image copies in a RECOVERY TOCOPY operation.

When an Inline COPY is taken by the REORG SHRLEVEL CHANGE utility, the contents of the image copy data set is different from the copy taken by LOAD. Online REORG takes an initial image copy during the RELOAD phase, then while it is applying changes from the log, it takes an incremental image copy and appends it onto the full image copy. This process is repeated during each log iteration as many times as needed to complete the online REORG. The end result is an image copy that has incremental copies added to the end of it, and therefore will contain duplicate pages.

This is not the same as an image copy produced using the MERGECOPY utility, which does not contain duplicates. The recover utility has been modified to handle these image copies, as it recognizes that the last version of the page is the valid one. The UNLOAD utility, when reading an image copy created by a REORG SHRLEVEL CHANGE utility, does not recognize these extra pages as duplicates and unloads all pages. This may lead to duplicate rows in the output data set. DSN1COPY and DSN1PRNT will produce error messages when inline copies are used with them if the new keyword INLCOPY is not specified.

### 9.1.2 Copying indexes

Starting with Version 6, DB2 has the ability to take full image copy of indexes. These copies can then be used by the RECOVER INDEX utility. The RECOVER utility recovers an index in the same manner as a table space, that is, restoring an image copy and then applying the log. This method allows for faster recovery of an index, assuming timely image copies are taken, when compared to rebuilding the index, and also it allows for the recovery of table spaces and indexes in parallel, thus reducing the outage required.

Indexes can be copied by using the option COPY YES, which is available in the CREATE/ALTER index DDL statements. COPY YES can be specified to indicate that the index can be copied by the COPY utility.

The COPY YES attribute:

- ▶ Allows DB2 full image copies and concurrent copies to be taken.
- ▶ Allows the use of the RECOVER utility on the index.
- ▶ Enables SYSCOPY recording for utility activity on the index.
- ▶ Enables SYSLGRNX recording for the index.
- ▶ Enables the setting of ICOPY and CHKP pending states for the index.
- ▶ Enables down-level detection on the index.
- ▶ Enables RECOVER from SYSTEM LEVEL BACKUP.

Altering the index to COPY NO prohibits and disables the above and also removes the recovery information from SYSCOPY and SYSLGRNX for the index.

The following restrictions apply to copying an index:

- ▶ CHANGELIMIT cannot be used.
- ▶ The copy data set is a sequential data set with a 4 KB LRECL.
- ▶ Inline image copies cannot be used.
- ▶ Incremental image copies cannot be used.
- ▶ DSNUM ALL must be used for NPI.
- ▶ RECOVER from SYSTEM LEVEL BACKUP could not be used.
- ▶ For compressed indexes, COPY creates an image copy of the index that is not compressed.

The index state, ICOPY (informational copy pending) indicates that a copy of the index should be taken and is set by:

- ▶ REBUILD INDEX
- ▶ REORG INDEX
- ▶ REORG TABLESPACE (LOG YES or LOG NO)
- ▶ LOAD TABLE (LOG YES or LOG NO)
- ▶ ALTER to padded
- ▶ ALTER to not padded
- ▶ ALTER add of a key column
- ▶ ALTER of a numeric data type key column

After the ICOPY state has been set, the index must be copied if the RECOVER utility is to be used against the index. If the copy is not taken, then in the event of a recovery, a REBUILD INDEX must be used to build the index from the underlying table. The ICOPY state does *not* prevent any processing against the table or the index. This state is also set on initial creation of the index with COPY YES and if the index is altered to be COPY YES.

The index state is CHKP (check pending). It indicates that the index *might* be out of step with the underlying table and that the CHECK INDEX utility should be run. This state can be set by an index point-in-time recovery. During the running of the check, the index is unavailable for any read or write activity.

The catalog table SYSIBM.SYSCOPY has a column named, OTYPE, which has the values for 'T' for table space and 'I' for index. The index name is recorded under the column TSNAME and there is a new value of 'B' for ICTYPE, which indicates REBUILD INDEX.

Other utilities have been enhanced to work with indexes that have COPY YES set:

- ▶ RECOVER
  - Support is included to recover from an image copy. This will end RC8 if COPY YES is not set, if an unrecoverable log event has occurred, or if there are no image copies for the index.
  - The index is handled in the same way as table space recoveries.
- ▶ REPORT
  - Indexes can now be specified.
  - Unusable image copies are marked, with (), to signify unsuitability.
  - The TABLESPACESET option reports all associated indexes.
- ▶ QUIESCE
  - SYSCOPY row, type 'Q', is inserted for table spaces and associated indexes.
  - Indexes quiesced are identified in the output job.
- ▶ MODIFY

MODIFY RECOVERY removes table space and associated index recovery information to keep the table space and index recovery information in sync.
- ▶ REBUILD INDEX
  - This sets ICOPY status.
  - This updates SYSCOPY with ICTYPE = 'B'.
- ▶ REORG INDEX
  - This updates SYSCOPY with ICTYPE = 'W'.
  - This sets ICOPY status.
- ▶ REPAIR
  - SET INDEX support is added to reset CHKP and ICOPY.
  - LEVELID support for indexes is provided.

Some other changes are these:

- ▶ DISPLAY DATABASE
  - A new ADVISORY option is provided to display indexes in ICOPY status.
  - RESTRICT displays indexes in CHKP status.
- ▶ START DATABASE

ACCESS(FORCE) can be used with indexes.
- ▶ DSN1COPY/DSN1PRNT

Option FULLCOPY supports index image copies.

## Recommendations

The following are recommendations for the image copying of indexes:

- ▶ Use image copy on large indexes with low update activity that require high availability.
- ▶ Copy indexes in conjunction with the underlying table space.
- ▶ Recover table spaces and their indexes to the same point-in-time to avoid CHKP being set.



### 9.1.3 Lists

The COPY utility has the ability to define lists, that enable a group of objects to be copied by the one COPY utility. This is different from the LISTDEF command, described in 3.1, “Wildcards” on page 48, in that the objects have to be explicitly coded in the COPY statement.

The utility processes the objects in the order that they are specified in the list. An example is shown in Example 9-1.

*Example 9-1 Specifying lists in the COPY utility*

---

```
COPY
  TABLESPACE DB1.TS1 COPYDDN (IC1,IC2) RECOVERYDDN(IC3,IC4)
  INDEX SYSADM1.IND1 COPYDDN(IC5,IC6) RECOVERYDDN(IC7,IC8)
```

---

The phases of COPY can now switch between REPORT and COPY because each table space in the list with a CHANGELIMIT specification (see 9.1.5, “The CHANGELIMIT option” on page 231) will have a REPORT phase.

If SHRLEVEL REFERENCE is specified, then DB2 drains the write claim class on ALL table/index spaces during the UTILINIT phase, and holds the claim until the last object in the list has been copied. The SYSCOPY row for all the objects in the list are written at the same time when the last object has completed, and the START\_RBA is the same for all rows, that being the RBA/LRSN after the last object has completed.

With SHRLEVEL CHANGE, DB2 establishes the read class claim on each object individually and releases the claim when it completes the object; it does not wait until every object has been completed. The SYSCOPY row for each object is inserted when the individual copy has completed and contains the START\_RBA pertinent to the RBA/LRSN at the start of the processing for that object.

If you specify OPTIONS EVENT(ITEMERROR,SKIP), each object in the list is placed in UTRW status and the read claim class is held only while the object is being copied. If you do not specify OPTIONS EVENT(ITEMERROR,SKIP), all of the objects in the list are placed in UTRW status and the read claim class is held on all objects for the entire duration of the COPY. The advantage is an UTRW state only for the duration of a copy of an individual page set at the cost of the serialization required for each pageset in the list to be processed.

When processing a list, if COPY encounters an error with one of the objects, it continues processing the next object in the list after issuing an error message. The return code at the end of the COPY utility will reflect the highest error set by the utility. To determine which objects caused the return code, the utility output will have to be used.

LISTDEF provides a dynamic method of building the LISTS and removes much of the LIST maintenance when new objects are created. Refer to Chapter 3, “Simplifying utilities with wildcarding and templates” on page 47 for details about LISTDEF.

## 9.1.4 COPY parallelism

The objective of the introduction of parallelism is to reduce the elapsed time of the COPY utility when run against a list of objects. To use parallelism with the COPY and RECOVER utilities, the keyword `PARALLEL n` has to be specified; this tells DB2 the optimum number of parallel threads that are to be processed. You can control the number of tape devices to allocate for the copy function by using `TAPEUNITS` parameter. If the keyword is omitted, then parallelism is not used. DB2 will use number 1 for `n` and uses three threads for a single-object COPY job. DB2 will also reduce the specified number of parallel tasks if there is a shortage of resources. If this is done, then message `DSNU397I` is issued, and the message `DSNU427I` will also be issued to identify the number of parallel tasks being used.

When using parallelism, COPY creates pairs of subtasks equivalent to the number of parallel tasks requested (or chosen by DB2). There are two subtasks per object: one to read the data and one to write the output. See Figure 9-1 for an example of `COPY PARALLEL(3)`. If the list contains more than three objects, then the subtasks are reused for subsequent objects in the list. To calculate the number of the threads needed when `PARALLEL` is specified, use this formula:

$$\text{Threads} = (n * 2 + 1)$$

The value *n* is the number of objects that should be processed in parallel, regardless of the total number of objects in the list.

Parallelism is supported for copies written to disk or tape. When you explicitly specify objects with the `PARALLEL` keyword, the objects are not necessarily processed in the specified order. Objects that are to be written to tape and whose file sequence numbers have been specified in the JCL are processed in the specified order. If templates are used, you cannot specify file sequence numbers. In the absence of overriding JCL specifications, DB2 determines the placement and the order of processing for such objects. When only templates are used, objects are processed according to their size, with the largest objects processed first.

Restart is only supported from the last commit point of each object processed in parallel.

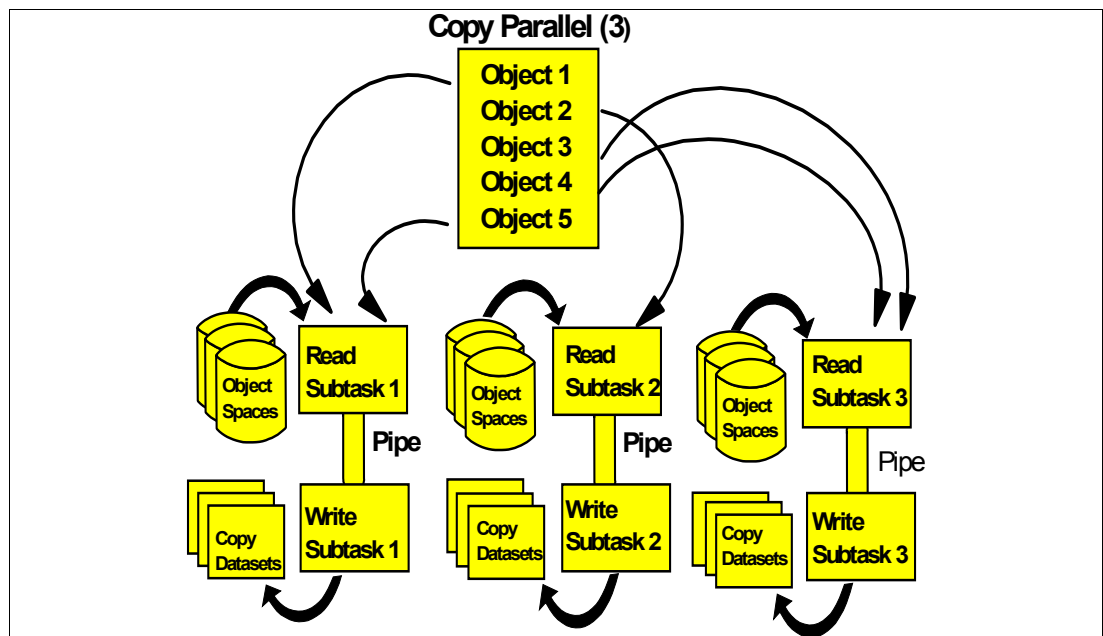


Figure 9-1 Parallel COPY with three subtasks

Example 9-2 show the statements required to build a COPY list using the LISTDEF functionality.

*Example 9-2 Parallel COPY using LISTDEF*

---

```

OPTIONS EVENT(ITEMERROR,SKIP)
  TEMPLATE LCOPY
    DSN(DB2V910G.&DB..&TS..T&TIME..P&PA.L)
    VOLUMES(SBOX59,SBOX60,SBOX58,SBOX57)
  LISTDEF DB01A
    INCLUDE TABLESPACE U9G01T11.TSLINEI PARTLEVEL
    INCLUDE INDEXSPACES TABLESPACE U9G01T11.TSLINEI
  COPY LIST DB01A FULL YES PARALLEL(4)
    SHRLEVEL REFERENCE
    COPYDDN(LCOPY)

```

---

Note that the partitioning index does not have the COPY YES parameter. It is still included in the list and produces an error message stating that the copy parameter is not set. In this case, the OPTIONS EVENT(ITEMERROR,SKIP) bypasses the error and COPY continues processing. The output from the job is shown in Example 9-3.

*Example 9-3 Output from Parallel COPY using LISTDEF*

---

```

1DSNU000I  DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = CPITEM
ODSNU050I  DSNUGUTC -  OPTIONS EVENT(ITEMERROR,SKIP)
DSNU1035I  DSNUIHDR - OPTIONS STATEMENT PROCESSED SUCCESSFULLY
ODSNU050I  DSNUGUTC -  TEMPLATE LCOPY DSN(DB2V910G.&DB..&TS..T&TIME..P&PA.L)
VOLUMES(SBOX59,SBOX60,SBOX58,SBOX57)
DSNU1035I  DSNUIHDR - TEMPLATE STATEMENT PROCESSED SUCCESSFULLY
ODSNU050I  DSNUGUTC -  LISTDEF DB01A INCLUDE TABLESPACE U9G01T11.TSLINEI PARTLEVEL INCLUDE
INDEXSPACES TABLESPACE
U9G01T11.TSLINEI
DSNU1035I  DSNUIHDR - LISTDEF STATEMENT PROCESSED SUCCESSFULLY
ODSNU050I  DSNUGUTC -  COPY LIST DB01A FULL YES PARALLEL(4) SHRLEVEL REFERENCE COPYDDN(LCOPY)
DSNU427I   DSNUBBID - OBJECTS WILL BE PROCESSED IN PARALLEL,
                  NUMBER OF OBJECTS = 4

DSNU425I   -DB2G DSNUBAII - INDEXSPACE U9G01T11.PXL#OKSD  DOES NOT HAVE THE COPY YES ATTRIBUTE
DSNU1027I   -DB2G DSNUBAII - PROCESSING CONTINUES DUE TO OPTIONS ITEMERROR SKIP
DSNU1038I   DSNUGDYN - DATASET ALLOCATED.  TEMPLATE=LCOPY
                  DDNAME=SYS00001
                  DSN=DB2V910G.U9G01T11.TSLINEI.T002257.P00001L
DSNU1038I   DSNUGDYN - DATASET ALLOCATED.  TEMPLATE=LCOPY
                  DDNAME=SYS00002
                  DSN=DB2V910G.U9G01T11.TSLINEI.T002257.P00002L
DSNU1038I   DSNUGDYN - DATASET ALLOCATED.  TEMPLATE=LCOPY
                  DDNAME=SYS00003
                  DSN=DB2V910G.U9G01T11.TSLINEI.T002257.P00003L
DSNU1038I   DSNUGDYN - DATASET ALLOCATED.  TEMPLATE=LCOPY
                  DDNAME=SYS00004
                  DSN=DB2V910G.U9G01T11.TSLINEI.T002257.P00000L
DSNU400I    DSNUBBID - COPY PROCESSED FOR INDEXSPACE U9G01T11.TESTNPI2
                  NUMBER OF PAGES=9751
                  AVERAGE PERCENT FREE SPACE PER PAGE = 0.00
                  PERCENT OF CHANGED PAGES = 0.00
                  ELAPSED TIME=00:00:22

```

```

DSNU1038I  DSNUGDYN - DATASET ALLOCATED.  TEMPLATE=LCOPY
                DDNAME=SYS00005
                DSN=DB2V910G.U9G01T11.TESTNPI.T002257.P00000L
DSNU400I   DSNUBBID - COPY PROCESSED FOR TABLESPACE U9G01T11.TSLINEI DSNUM 1
                NUMBER OF PAGES=13595
                AVERAGE PERCENT FREE SPACE PER PAGE = 3.34
                PERCENT OF CHANGED PAGES = 0.00
                ELAPSED TIME=00:00:27
DSNU400I   DSNUBBID - COPY PROCESSED FOR TABLESPACE U9G01T11.TSLINEI DSNUM 2
                NUMBER OF PAGES=13831
                AVERAGE PERCENT FREE SPACE PER PAGE = 3.35
                PERCENT OF CHANGED PAGES = 0.00
                ELAPSED TIME=00:00:28
DSNU400I   DSNUBBID - COPY PROCESSED FOR INDEXSPACE U9G01T11.TESTNPI
                NUMBER OF PAGES=10128
                AVERAGE PERCENT FREE SPACE PER PAGE = 0.00
                PERCENT OF CHANGED PAGES = 0.00
                ELAPSED TIME=00:00:13
DSNU400I   DSNUBBID - COPY PROCESSED FOR TABLESPACE U9G01T11.TSLINEI DSNUM 3
                NUMBER OF PAGES=107667
                AVERAGE PERCENT FREE SPACE PER PAGE = 3.36
                PERCENT OF CHANGED PAGES = 0.00
                ELAPSED TIME=00:01:22
DSNU428I   DSNUBBID - DB2 IMAGE COPY SUCCESSFUL FOR TABLESPACE U9G01T11.TSLINEI DSNUM 1
DSNU428I   DSNUBBID - DB2 IMAGE COPY SUCCESSFUL FOR TABLESPACE U9G01T11.TSLINEI DSNUM 2
DSNU428I   DSNUBBID - DB2 IMAGE COPY SUCCESSFUL FOR TABLESPACE U9G01T11.TSLINEI DSNUM 3
DSNU428I   DSNUBBID - DB2 IMAGE COPY SUCCESSFUL FOR INDEXSPACE U9G01T11.TESTNPI2
DSNU428I   DSNUBBID - DB2 IMAGE COPY SUCCESSFUL FOR INDEXSPACE U9G01T11.TESTNPI
DSNU012I   DSNUGBAC - UTILITY EXECUTION TERMINATED, HIGHEST RETURN CODE=8

```

---

The number of parallel tasks is set to 4 and the number of objects available for copying is 5 (ignoring the rejected partitioning index). The output from a -DISPLAY UTIL command in Example 9-4 shows that there were four pairs of read/write subtasks running in parallel.

*Example 9-4 -DISPLAY UTILITY for Parallel COPY*

---

```

-db2a dis util(*)
DSNU105I  -DB2G DSNUGDIS - USERID = DB2R7
                MEMBER =
                UTILID = CPITEM
                PROCESSING UTILITY STATEMENT 1
                UTILITY = COPY
                PHASE = COPY  COUNT = 0
                NUMBER OF OBJECTS IN LIST = 6
                LAST OBJECT STARTED = 0
                STATUS = ACTIVE
DSNU111I  -DB2G DSNUGDIS - SUBPHASE = COPYW COUNT = 0
DSNU111I  -DB2G DSNUGDIS - SUBPHASE = COPYR COUNT = 0
DSNU111I  -DB2G DSNUGDIS - SUBPHASE = COPYW COUNT = 0
DSNU111I  -DB2G DSNUGDIS - SUBPHASE = COPYR COUNT = 0
DSNU111I  -DB2G DSNUGDIS - SUBPHASE = COPYW COUNT = 0
DSNU111I  -DB2G DSNUGDIS - SUBPHASE = COPYR COUNT = 0
DSNU111I  -DB2G DSNUGDIS - SUBPHASE = COPYR COUNT = 0
DSNU111I  -DB2G DSNUGDIS - SUBPHASE = COPYW COUNT = 0
DSNU9022I -DB2G DSNUGCCC '-DIS UTIL' NORMAL COMPLETION

```

---

Also worth noting is that the number of objects in the list is six (consisting of three table space partitions, one partitioning index, and two NPIs), and that the partitioning index is included and later rejected by the COPY utility.

### 9.1.5 The CHANGELIMIT option

CHANGELIMIT is responsible for automating the decision making process of whether to image copy and whether to take incremental or full copies.

There are two modes to the CHANGELIMIT: a report-only mode and a run mode. If REPORTONLY is specified, DB2 will only execute the REPORT phase of the utility and will report on the following values:

- ▶ The total number of pages. This is the number of pages in the table space and may include preformatted pages. It is the number of pages that a full copy will copy.
- ▶ Empty pages in a segmented table space.
- ▶ Number of changed pages, that is, the number of pages an incremental copy would copy.
- ▶ Percentage of changed pages.
- ▶ Recommendation of the type of image copy required.

The report for table spaces will be displayed at the following levels:

- ▶ Partition level, if the table space is partitioned.
- ▶ Data set level, if the table space is not partitioned.
- ▶ The entire table space.

The recommendation given by the REPORT phase is based on values provided by the CHANGELIMIT parameter values. These values specify the range of changed pages expressed as a percentage of total pages in the table space.

There are three options that can be supplied by the CHANGELIMIT parameter:

- ▶ ANY  
Specifies that COPY is to take a full image copy if any pages have changed since the last image copy. This option is not using any percentage value.
- ▶ One value
  - An incremental image copy is recommended if the percentage of changed pages is between zero and the supplied value.
  - A full image copy is recommended if the percentage of changed pages is greater than the supplied value.
  - No image copy is recommended if no pages have been changed.
- ▶ Two values (v1,v2)
  - An incremental is recommended if the percentage of changed pages is greater than v1 and less than v2.
  - A full copy is recommended if the percentage of changed pages is greater than v2.
  - No image copy is recommended if the percentage of changed pages is less than v1.

The default values are 1,10. Example 9-5 illustrates the utility stream input to check for image copies using the range 0 to 25.

*Example 9-5 Specifying CHANGELIMIT*

---

```
COPY LIST DB01A  CHANGELIMIT(0,25)
      SHRLEVEL REFERENCE
      COPYDDN(LOCALDDN)
```

---

Example 9-6 shows the report from the COPY command.

*Example 9-6 Report output from COPY utility*

---

DBNAME	TSNAME	DSNUM	4KB PAGES	CHANGED PAGES	PERCENT OF CHANGED PAGES	ICTYPE
-----	-----	----	-----	-----	-----	-----
U9G01T11	TSLINEI	1	3,415	2	0.05	
U9G01T11	TSLINEI	2	3,479	0	0.00	
U9G01T11	TSLINEI	3	26,938	0	0.00	
U9G01T11	TSLINEI	ALL	33,832	2	< 0.01	I

---

The report shows that the COPY takes an incremental copy at the DSNUM ALL level.

COPY sets a return code dependent upon which option is taken:

- ▶ RC1 if the utility does not take an image copy
- ▶ RC2 if an incremental copy is taken
- ▶ RC3 if a full copy is taken
- ▶ RC8 if a broken page has been encountered

**Note:** Image copy data sets are always allocated by COPY, even if REPORTONLY is specified. Care should be exercised when using GDGs to ensure that valid image copies are not rolled out. This is the case whether specifying a DD statement or using Templates.

This enables the job stream to automate the processing of image copies, that is, a REPORTONLY STEP is run and a subsequent step processes a full copy or an incremental copy, depending upon the return code. If using this method, ensure that the SYSCOPY DDNAME in the REPORTONLY step is set to DUMMY to ensure that extra data sets are not cataloged.

**Note:** CHANGELIMIT opens each data set to determine whether an image copy is required or not. CHANGELIMIT will recall the data sets if migrated (if AUTORECALL is allowed). CHANGELIMIT will update the last referenced date on any data set it opens, so data sets will not get migrated if CHANGELIMIT is used frequently.

## 9.1.6 The CHECKPAGE option

The COPY utility checks the integrity of data pages as it is copying the table space or index space. This option is equivalent to the checking undertaken by DSN1COPY with CHECK. In DB2 9, this option is a default for TABLESPACE, but not for INDEXES, and DB2 will validate each page of the table space or index page as it is processed.

The benefit of this option is that any errors in the pages are reported at the time that the backup is taken and not when the image copy is required by the RECOVER utility. The risk of there being any errors when taking the image is low, but the impact of finding them at recovery time is high.

DB2 9 no longer sets copy pending when the utility finds a broken page. The utility issues the message that describes the type of error (DSNU518I), and it continues checking pages, but not copying anything: Just one error message is issued for one page. Upon completion, COPY updates the SYSIBM.SYSCOPY table in a new column TTYPE with value of 'B'. With this entry, DB2 will prevent subsequent incremental image copies from been taken because the previous full copy has not copied the full table space to the image copy.

If COPY identifies an invalid page, it will:

- ▶ Identify the page as broken.
- ▶ Issue message DSNU518I if the page is a data page.
- ▶ Issue message DSNU441I if the page is a space map page.
- ▶ Complete RC8.

Then you have to look for the COPY utility job that has return code RC=8 to fix all the found pages as soon as possible.

Remedial action should be initiated immediately by using either RECOVER PAGE (page numbers are given by DSNT518I and DSNU441I), RECOVER, or REPAIR.

Another improvement that comes with DB2 9 is that the CPU is reduced due to changes in buffering from Least Recently Used (LRU) to Most Recently Used (MRU). This should also reduce the impact that the COPY utility has on the buffer pool in regards to applications using the same buffer pool.

### 9.1.7 The SCOPE parameter

The SCOPE parameter has the following options:

- ▶ ALL (default)

This option will copy all objects without regard to the pending state.

- ▶ PENDING

SCOPE PENDING was created to have the ability to copy just the objects in copy pending or informational copy pending status. For partitioned objects, you have to provide a list of partitions that is in copy pending or informational copy pending status if you want just copy them. You can use this option by using LISTDEF and TEMPLATE. If a space level copy (for example, DSNUM ALL) is requested, if any of the partitions are in COPY pending status, then an image copy of the entire table space or indexspace is created.

Example 9-7 shows four table spaces to be copied, but only two are in the COPY pending status.

*Example 9-7 Source and output for SCOPE PENDING*

---

```
//DB2R7001 JOB CLASS=A,TIME=1440,REGION=0M,MSGCLASS=T,MSGLEVEL=(1,1)
/*JOBPARM S=SC63
//COPY1 EXEC DSNUPROC,SYSTEM=DB9A,UID=''
//DSNUPROC.SYSIN DD *
TEMPLATE COPY_TS DSN &DB..&TS..&IC.D&DA..T&TI.
UNIT=SYSDA
COPY TABLESPACE DB2PEDB.DB2PMFAC COPYDDN(COPY_TS) SCOPE PENDING
COPY TABLESPACE DSN8D91P.DSN8S91C COPYDDN(COPY_TS) SCOPE PENDING
COPY TABLESPACE ADHTOOLS.ADHTSHRU COPYDDN(COPY_TS) SCOPE PENDING
COPY TABLESPACE DB2PEDB.DB2P11G5 COPYDDN(COPY_TS) SCOPE PENDING
```

12:03:02.71 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = DB2R7.DB2R7001  
12:03:02.75 DSNUGTIS - PROCESSING SYSIN AS EBCDIC  
12:03:02.76 DSNUGUTC - TEMPLATE COPY\_TS DSN &DB..&TS..&IC.D&DA..T&TI. UNIT=SYSD  
12:03:02.76 DSNUJTDR - TEMPLATE STATEMENT PROCESSED SUCCESSFULLY  
12:03:02.76 DSNUGUTC - COPY TABLESPACE DB2PEDB.DB2PMFAC COPYDDN(COPY\_TS) SCOPE  
12:03:02.85 DSNUGDYN - DATASET ALLOCATED. TEMPLATE=COPY\_TS  
DDNAME=SYS00001  
DSN=DB2PEDB.DB2PMFAC.FD01.T160325  
12:03:03.77 DSNUBBID - COPY PROCESSED FOR TABLESPACE DB2PEDB.DB2PMFAC  
NUMBER OF PAGES=6454  
AVERAGE PERCENT FREE SPACE PER PAGE = 45.96  
PERCENT OF CHANGED PAGES = 0.06  
ELAPSED TIME=00:00:00  
274 12:03:03.79 DSNUBAFI - DB2 IMAGE COPY SUCCESSFUL FOR TABLESPACE DB2PEDB.DB2P  
12:03:03.79 DSNUGUTC - COPY TABLESPACE DSN8D91P.DSN8S91C COPYDDN(COPY\_TS) SCOPE  
274 12:03:03.79 DSNUBAII - **SCOPE PENDING WAS SPECIFIED, BUT NO OBJECTS WITH  
PENDING STATUS WERE FOUND**  
12:03:03.80 DSNUGUTC - COPY TABLESPACE ADHTOOLS.ADHTSHRU COPYDDN(COPY\_TS) SCOPE  
274 12:03:03.80 DSNUBAII - **SCOPE PENDING WAS SPECIFIED, BUT NO OBJECTS WITH  
PENDING STATUS WERE FOUND**  
12:03:03.81 DSNUGUTC - COPY TABLESPACE DB2PEDB.DB2P11G5 COPYDDN(COPY\_TS) SCOPE  
12:03:03.90 DSNUGDYN - DATASET ALLOCATED. TEMPLATE=COPY\_TS  
DDNAME=SYS00002  
DSN=DB2PEDB.DB2P11G5.FD01.T160326  
12:03:03.93 DSNUBBID - COPY PROCESSED FOR TABLESPACE DB2PEDB.DB2P11G5  
NUMBER OF PAGES=6  
AVERAGE PERCENT FREE SPACE PER PAGE = 7.33  
PERCENT OF CHANGED PAGES = 16.66  
ELAPSED TIME=00:00:00  
274 12:03:03.93 DSNUBAFI - DB2 IMAGE COPY SUCCESSFUL FOR TABLESPACE DB2PEDB.DB2P  
12:03:03.94 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0

---



## 9.1.8 The CLONE option

With this option, the COPY utility will copy just CLONE TABLE or INDEX data. If you are using the LIST keyword to specify a list of objects, COPY will process only clone table or indexes on clone tables. In Example 9-8, you can see how COPY works when using the CLONE option. There are two table spaces to be copied, but just one has a clone table; then the processing skipped message appears.

### *Example 9-8 Using CLONE option*

---

```
//COPY1 EXEC DSNUPROC,SYSTEM=DB9A,UID=' '
//DSNUPROC.SYSIN DD *
TEMPLATE COPY_TS DSN &DB..&TS..&IC.D&DA..T&TI.
                UNIT=SYSDA
LISTDEF COMPLIST
                INCLUDE  TABLESPACE DB1.TS1
                INCLUDE  TABLESPACE DB2PEDB.DB2PMFAC
COPY LIST COMPLIST COPYDDN(COPY_TS) CLONE
```

12:29:42.35 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = DB2R7.DB2R7001  
12:29:42.39 DSNUGTIS - PROCESSING SYSIN AS EBCDIC  
12:29:42.39 DSNUGUTC - TEMPLATE COPY\_TS DSN &DB..&TS..&IC.D&DA..T&TI. UNIT=SYSD  
12:29:42.39 DSNUJTDR - TEMPLATE STATEMENT PROCESSED SUCCESSFULLY  
12:29:42.40 DSNUGUTC - LISTDEF COMPLIST INCLUDE TABLESPACE DB1.TS1 INCLUDE TABL

12:29:42.40 DSNUILD - LISTDEF STATEMENT PROCESSED SUCCESSFULLY  
12:29:42.40 DSNUGUTC - COPY LIST COMPLIST COPYDDN(COPY\_TS) CLONE  
12:29:42.41 DSNUGULM - **PROCESSING LIST ITEM: TABLESPACE DB1.TS1**  
12:29:42.41 DSNUGULM - **PROCESSING SKIPPED FOR TABLESPACE DB2PEDB.DB2PMFAC REASON**  
**CODE=1**  
12:29:42.46 DSNUGDYN - DATASET ALLOCATED. TEMPLATE=COPY\_TS  
DDNAME=SYS00001  
DSN=DB1.TS1.FD01.T163005  
12:29:42.51 DSNUBBID - COPY PROCESSED FOR TABLESPACE DB1.TS1  
NUMBER OF PAGES=3  
AVERAGE PERCENT FREE SPACE PER PAGE = 32.33  
PERCENT OF CHANGED PAGES = 1.66  
PERCENT OF CHANGED PAGES = 1.66  
ELAPSED TIME=00:00:00  
274 12:29:42.52 DSNUBAFI - DB2 IMAGE COPY SUCCESSFUL FOR TABLESPACE DB1.TS1  
12:29:42.53 DSNUGULM - **PROCESSING SKIPPED FOR 1 OF 2 OBJECTS**  
12:29:42.53 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0

---

## 9.1.9 Full or incremental copies

The two types of COPY are the full and incremental copy. The keyword FULL YES requests a copy of all the pages in the table space or index space. The keyword FULL NO is used to copy only the pages that have been changed since the last full copy or incremental image copy. The incremental copy is not available for index copies.

With the current performance of COPY, the recommended limit for incremental is 50%. To identify the percentage of changed pages that exist in a data set, use the CHANGELIMIT REPORTONLY option of COPY, as discussed in 9.1.5, “The CHANGELIMIT option” on page 231.

You can take incremental image copies if the following conditions apply:

- ▶ A full image copy of the table space exists.
- ▶ The COPY-pending status is not on for that table space.
- ▶ The last copy was taken without the CONCURRENT option.

When using incremental copy, one point to remember is that to improve recovery time, MERGECOPY should be run to incorporate the incremental copies and the full copy into a new full copy, which will improve the time taken by the RECOVER utility. When running MERGECOPY, DB2 will request that all the incremental copies and the full copy be available at the same time. This may cause a problem if the copies are on tape drives and there are more copies than drives available. Therefore, MERGECOPY should be run frequently, especially for copies taken to tape, but also for all media to speed up the recovery process.

### 9.1.10 SHRLEVEL REFERENCE or SHRLEVEL CHANGE

The option to take a copy with SHRLEVEL REFERENCE or SHRLEVEL CHANGE will depend upon the required availability of the table space to applications and users.

IF SHRLEVEL REFERENCE is specified, applications can only read the data and not update the data. This may be contrary to business requirements that demand higher availability. To help meet this need, the use of SHRLEVEL CHANGE should be used; this will allow full read and write access during the copy process.

Some utilities, for example, REORG, demand that a SHRLEVEL REFERENCE copy be taken after the utility has completed, which led to the outage from REORGs being extended. REORG can take inline image copies, which are SHRLEVEL REFERENCE copies, during execution, which eliminates the delay in availability, as discussed in 9.1.1, “Inline COPY with LOAD and REORG” on page 224.

### 9.1.11 Tape or disk

Image copies can be written to either tape or disk. The question is: which one?

The advantage of image copy to disk is that the elapsed time of the copy is less, due to the speed of the media. Refer to 9.1.4, “COPY parallelism” on page 228 for more information. The number of tape units may also restrict the running of multiple copy jobs and parallelism. The preferred choice is writing to disk.

The secondary, or recovery, image copies have traditionally been sent to tape, or any remote device, while sending the primary image copy to disk, by specifying the appropriate unit in the RECOVDDN option. This can drastically reduce the benefits of writing the primary copy to disk as the elapsed time of the utility is determined by the slowest device or narrowest bandwidth in case of transmission. With the COPYTOCOPY utility (refer to 9.2, “COPYTOCOPY” on page 237), this delay can be eliminated and the secondary or recovery copy initiated asynchronously, thus allowing higher availability of the application.

Other advantages of using disk over tape are:

- ▶ Faster elapsed time
- ▶ Faster recovery times, due to speed of the media
- ▶ Faster MERGECOPY, as well as the removal of problems, with a number of drives available for incremental copies and the same tape being used for full and incremental copies

With the cost of new disks coming down and the new disk technology, we recommend that image copies be taken to disk and the new utility COPYTOCOPY be used to take recovery backups, or all should be taken to disk and then migrated with an HSM type of function.

## 9.2 COPYTOCOPY

The COPY utility and the LOAD and REORG utilities with Inline COPY can produce local primary and backup copies and recovery site primary and backup copies. Taking two local and two recovery sites copies all at once can tie up four tape drives per utility, and if multiple copies are running concurrently, then there may be a need for more tape drives than are available to the site. In addition, some customers use remote attached tape drives for their recovery site copies, thus causing the copy to take longer and decreasing data availability because of bandwidth constraints. The requirement, therefore, is to be able to take a single copy and later make additional copies, asynchronously from the original, and record the copies in SYSCOPY, as shown in Figure 9-2. This will increase the data availability of objects, due to reducing the elapsed time, and will also reduce the costs of running the COPY utility.

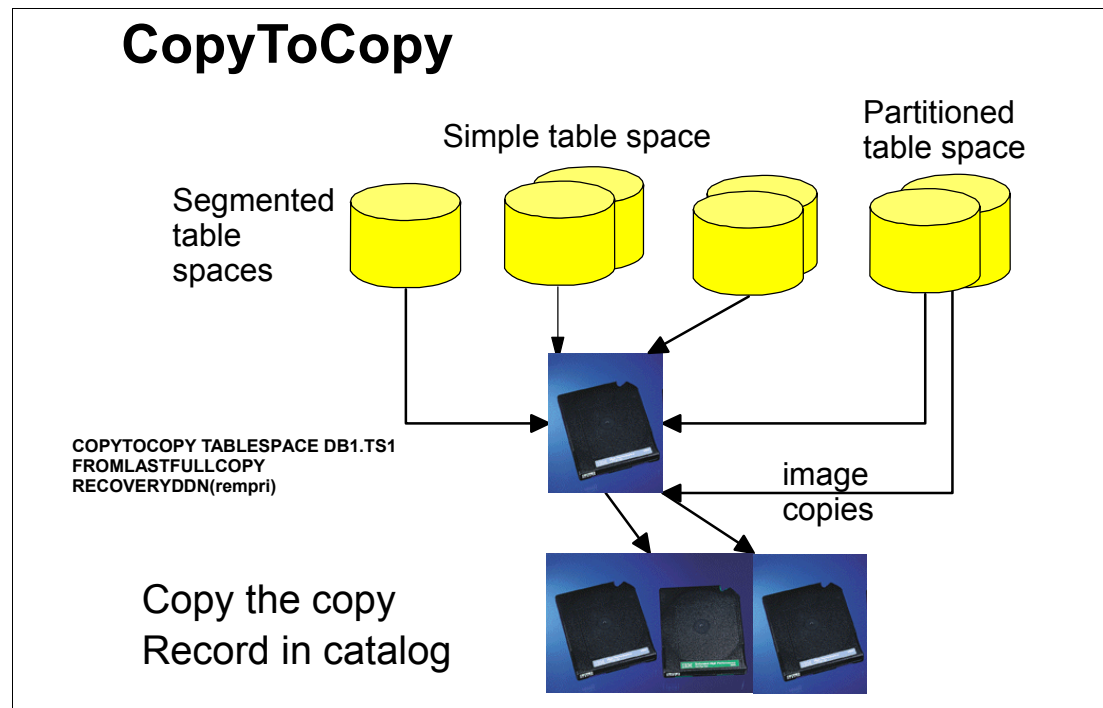


Figure 9-2 COPYTOCOPY

COPYTOCOPY can be used to make copies from an image copy that was taken by the COPY utility, including inline copies taken by REORG and LOAD utilities. COPYTOCOPY will leave the target object in read write access (UTRW), which allows for most other utilities and SQL statements to run concurrently on the same target objects. The utilities that are not allowed to run concurrently with COPYTOCOPY on the same target object are utilities that insert or delete records into SYSCOPY. They are:

- ▶ COPY
- ▶ LOAD
- ▶ MERGECOPY
- ▶ MODIFY
- ▶ RECOVER
- ▶ REORG

The only other utilities not allowed concurrently are utilities that operate with SYSCOPY as their target.

Input to the utility is either the local primary or the recovery site primary copy from which COPYTOCOPY can make up to three copies of one or more of the following types:

- ▶ Local primary
- ▶ Local backup
- ▶ Recovery site primary
- ▶ Recovery site backup

COPYTOCOPY does not support the following:

- ▶ DSNDB01.SYSUTILX and its indexes
- ▶ DSNDB01.DBD01 and its indexes
- ▶ DSNDB06.SYSCOPY and its indexes
- ▶ Image copies taken with CONCURRENT option
- ▶ Image copies taken by REORG part range

COPYTOCOPY does not check recoverability of an object.

The copies created by COPYTOCOPY can be used by RECOVER when recovering a table space and are also available to MERGECOPY, UNLOAD, and another COPYTOCOPY.

Output from the utility is up to three “new” image copies and rows in SYSCOPY that describe the image copies. All entries in SYSCOPY are the same as the original copy, apart from ICBACKUP, which describes the type of image copy, for example, remote primary, and so on.

**Note:** The TIMESTAMP field in SYSCOPY is the timestamp of the original image copy. If date and time are used in the data set name, as in Example 9-9, these values will contain the date and time that the COPYTOCOPY was run and not the date and time in the input copy data set.

COPYTOCOPY is recommended for use when:

- ▶ Additional copies of objects that are high availability are required and the extra copies are taken to slower devices.
- ▶ The target object is required to remain in UTRW status, allowing SQL statements to run concurrently with the same target object.

## 9.2.1 COPYTOCOPY options

COPYTOCOPY can use the LISTDEF and Template functionality described in Chapter 3, “Simplifying utilities with wildcarding and templates” on page 47. An example of this is shown in Example 9-9. This example also illustrates the statements required to take three copies from the local primary copy. It is not possible to use COPYTOCOPY to take a copy of the local primary and create another local primary copy.

*Example 9-9 COPYTOCOPY and LISTDEFs*

---

```
TEMPLATE LOCALDDN
  DSN(DB2V910G.&DB..&TS..D&DATE..T&TIME.&LOCREM.)
  VOLUMES(SBOX60)
TEMPLATE RECOVDDN
  DSN(DB2V910G.&DB..&TS..D&DATE..T&TIME.&LOCREM.)
  VOLUMES(SBOX60)
TEMPLATE RECOVDD2
  DSN(DB2V910G.&DB..&TS..D&DATE..T&TIME.Z)
  VOLUMES(SBOX60)
LISTDEF DB01A
  INCLUDE TABLESPACE U9G01T11.TSPSUPP1
COPYTOCOPY LIST DB01A FROMLASTCOPY
  COPYDDN(,LOCALDDN)
  RECOVERYDDN(RECOVDDN,RECOVDD2)
```

---

Options for identifying the copy to use as input are:

- ▶ **FROMLASTCOPY**  
Specifies the most recent image copy that was taken for the table space or index space to be the input to the COPYTOCOPY utility. This could be a full image copy or incremental copy retrieved from SYSIBM.SYSCOPY.
- ▶ **FROMLASTFULLCOPY**  
Specifies the most recent full image copy that was taken for the object to be the input to COPYTOCOPY job.
- ▶ **FROMLASTINRCOPY**  
Specifies the most recent incremental image copy that was taken for the object to be the input to COPYTOCOPY job. If specified with an INDEX or INDEX SPACE, this option will revert to FROMLASTFULLCOPY.
- ▶ **FROMCOPY *dsn***  
Specifies a particular image copy data set (*dsn*) as the input to COPYTOCOPY job. This option is not valid for LIST. The GDG names must be specified fully.

The output from Example 9-9 on page 239 is listed in Example 9-10.

*Example 9-10 COPYTOCOPY using FROMLASTCOPY*

---

```

DSNU000I  DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = PAOLR2
DSNU050I  DSNUGUTC - TEMPLATE LOCALDDN DSN(DB2V910G.&DB..&TS..D&DATE..T&TIME.&LOCREM.) VOLUMES(SBOX60)
DSNU1035I DSNUJTDR - TEMPLATE STATEMENT PROCESSED SUCCESSFULLY
DSNU050I  DSNUGUTC - TEMPLATE RECOVDDN DSN(DB2V910G.&DB..&TS..D&DATE..T&TIME.&LOCREM.) VOLUMES(SBOX60)
DSNU1035I DSNUJTDR - TEMPLATE STATEMENT PROCESSED SUCCESSFULLY
DSNU050I  DSNUGUTC - TEMPLATE RECOVDD2 DSN(DB2V910G.&DB..&TS..D&DATE..T&TIME.Z) VOLUMES(SBOX60)
DSNU1035I DSNUJTDR - TEMPLATE STATEMENT PROCESSED SUCCESSFULLY
DSNU050I  DSNUGUTC - LISTDEF DB01A INCLUDE TABLESPACE U9G01T11.TSPSUPP1
DSNU1035I DSNUIldr - LISTDEF STATEMENT PROCESSED SUCCESSFULLY
DSNU050I  DSNUGUTC - COPYTOCOPY LIST DB01A FROMLASTCOPY COPYDDN(,LOCALDDN)
RECOVERYDDN(RECOVDDN,RECOVDD2)
DSNU1038I DSNUGDYN - DATASET ALLOCATED.  TEMPLATE=LOCALDDN
                        DDNAME=SYS00001
                        DSN=DB2V910G.U9G01T11.TSPSUPP1.D2001151.T233937L
DSNU1038I DSNUGDYN - DATASET ALLOCATED.  TEMPLATE=RECOVDDN
                        DDNAME=SYS00002
                        DSN=DB2V910G.U9G01T11.TSPSUPP1.D2001151.T233937R
DSNU1038I DSNUGDYN - DATASET ALLOCATED.  TEMPLATE=RECOVDD2
                        DDNAME=SYS00003
                        DSN=DB2V910G.U9G01T11.TSPSUPP1.D2001151.T233937Z
DSNU1403I DSNU2BCC - LOCAL SITE PRIMARY DATA SET DB2V910G.U9G01T11.TSPSUPP1.D2001151.T233722L WITH
                        START_RBA 000058B794BC IS IN USE BY COPYTOCOPY
                        FOR TABLESPACE U9G01T11.TSPSUPP1
DSNU1404I DSNU2BDR - COPYTOCOPY PROCESSING COMPLETED FOR
                        TABLESPACE U9G01T11.TSPSUPP1
                        ELAPSED TIME = 00:00:38
                        NUMBER OF PAGES COPIED=24805
DSNU1406I DSNU2BDR - COPYTOCOPY COMPLETED. ELAPSED TIME = 00:00:38
DSNU010I  DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0

```

---

COPYTOCOPY will not allow the copying of an image copy, for example, local primary to another type, or, for example, remote primary when an image copy of the same type with the same LRSN already exists, that is, a remote primary was taken at the same time as the local primary was taken. The utility does not support inline copies made by the REORG with part range. To copy these copies, use the individual DSNUM(n), because COPYTOCOPY copies only the specified partition into the output data sets.

## 9.3 Concurrent COPY

This option of the COPY utility enables you to make full image copies using DFSMS Concurrent COPY. The copy process is initiated by the DB2 COPY utility when you specify the CONCURRENT keyword on the COPY statement. The image copy is recorded in SYSCOPY with ICTYPE = F and STYPE = C, for example, node "I00012" and STYPE = J for instance node "J0001".

To use COPY to take DFSMS concurrent copies, you need the following hardware:

- ▶ 3990 Model 3 or 3990 Model 6 controller at the extended platform attached to the disk. A COPY job fails if one or more of the table spaces are on a disk that does not have the right storage controller.
- ▶ Any DASD supporting FlashCopy capability.

In Example 9-11, you see the error messages you will get if you are using this option without the appropriate hardware and software support.

*Example 9-11 Output job for Concurrent COPY with error*

---

```

DSNU050I  DSNUGUTC - COPY TABLESPACE U9G01T11.TSPSUPP2 COPYDDN(MARCELO) DSNUM 3 CONCURRENT
DSNU1038I  DSNUGDYN - DATASET ALLOCATED.  TEMPLATE=MARCELO
                DDNAME=SYS00005
                DSN=PAOLOR4.UTIL.TSPSUPP2.P00003.T233225
DSNU421I  DSNUBBCM - START OF DFSMS MESSAGES
PAGE 0001  5695-DF175 DFSMSDSS V2R10.0 DATA SET SERVICES    2009.293 19:32
PARALLEL
ADR101I (R/I)-RI01 (01), TASKID 001 HAS BEEN ASSIGNED TO COMMAND 'PARALLEL'
DUM OPT(3) DATAS(INCL(PAOLOR1.DSNDBC.U9G01T11.TSPSUPP2.J0001.A003)) -
CAN CONC SHA TOL(ENQF) WAIT(0,0) -
OUTDD(SYS00005)
ADR101I (R/I)-RI01 (01), TASKID 002 HAS BEEN ASSIGNED TO COMMAND 'DUM'
ADR109I (R/I)-RI01 (01), 2009.293 19:32:33 INITIAL SCAN OF USER CONTROL STATEMENTS COMPLETED.
ADR014I (SCH)-DSSU (02), 2009.293 19:32:33 ALL PREVIOUSLY SCHEDULED TASKS COMPLETED. PARALLEL MODE
NOW IN EFFECT
ADR050I (002)-PRIME(01), DFSMSDSS INVOKED VIA APPLICATION INTERFACE
ADR016I (002)-PRIME(01), RACF LOGGING OPTION IN EFFECT FOR THIS TASK
ADR006I (002)-STEND(01), 2009.293 19:32:33 EXECUTION BEGINS
ADR411W (002)-DTDSC(04), DATA SET PAOLOR1.DSNDBC.U9G01T11.TSPSUPP2.J0001.A003 IN CATALOG
UCAT.VSBOX01 ON VOLUME SBOX58
                WAS NOT SERIALIZED ON REQUEST
ADR356E (002)-UIMXT(01), TASK TERMINATED BY UIM EXIT (24)
ADR735W (002)-TOMI (05), 2009.293 19:32:33 USE OF CONCURRENT COPY FAILED FOR DATA SET
                PAOLOR1.DSNDBC.U9G01T11.TSPSUPP2.J0001.A003 ON VOLUME SBOX59, 002.
SERIALIZATION WILL BE HELD AND CONCURRENT COPY WILL NOT BE USED.
ADR356E (002)-UIMXT(01), TASK TERMINATED BY UIM EXIT (23)
ADR801I (002)-DTDSC(01), DATA SET FILTERING IS COMPLETE. 1 OF 1 DATA SETS WERE SELECTED: 0 FAILED
SERIALIZATION AND 0
                FAILED FOR OTHER REASONS.
ADR006I (002)-STEND(02), 2009.293 19:32:33 EXECUTION ENDS
ADR013I (002)-CLTSK(01), 2009.293 19:32:33 TASK COMPLETED WITH RETURN CODE 0008
ADR012I (SCH)-DSSU (01), 2009.293 19:32:33 DFSMSDSS PROCESSING COMPLETE. HIGHEST RETURN CODE IS 0008
FROM:
                TASK    002

DSNU422I  DSNUBBCM - END OF DFSMS MESSAGE
DSNU409I  DSNUBBUM - NO HARDWARE SUPPORT FOR TABLESPACE U7G01T11.TSPSUPP2 DSNUM 3
DSNU012I  DSNUGBAC - UTILITY EXECUTION TERMINATED, HIGHEST RETURN CODE=8

```

---

If you use the CONCURRENT option:

- ▶ You must supply either a COPYDDN ddname, a RECOVERYDDN ddname, or both.
- ▶ If the SYSPRINT DD card points to a data set, you must use a DSSPRINT DD card.
- ▶ You must use the SHRLEVEL REFERENCE option for table spaces with 8 KB, 16 KB, or 32 KB page size unless the CI size for the object equals the DB2 page size (install panel DSNTIP7).

**Consideration:**

- ▶ You cannot use a copy made with DFSMS Concurrent COPY with the PAGE or ERRORRANGE options of the RECOVER utility.
- ▶ You cannot run the following DB2 stand-alone utilities on copies made by DFSMS Concurrent COPY:
  - DSN1COMP, DSN1COPY, and DSN1PRNT.
  - You can manually restore the DFSMS copy data set under a different name and run these utilities against the restored data set.

- ▶ If you try to take an incremental image copy of a table space after a full image copy is taken using DFSMS Concurrent COPY, you will receive the DSNU402I message, as shown in Example 9-12. Notice that the percent of changed pages is zero.

*Example 9-12 Output from incremental copy after a Full Concurrent COPY*


---

```

DSNU000I  DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = TEMP
DSNU050I  DSNUGUTC - TEMPLATE MARCELO UNIT SYSDA DSN(PAOLOR4.&STEPNAME..&SN..P&PA..T&TIME.)
DISP(MOD,CATLG,
CATLG)
DSNU1035I DSNUJTDR - TEMPLATE STATEMENT PROCESSED SUCCESSFULLY
DSNU050I  DSNUGUTC - COPY TABLESPACE U9G01T11.TSPSUPP2 COPYDDN(MARCELO) DSNUM 1 FULL NO
DSNU1038I DSNUGDYN - DATASET ALLOCATED. TEMPLATE=MARCELO
                DDNAME=SYS00001
                DSN=PAOLOR4.UTIL.TSPSUPP2.P00001.T003531
DSNU402I  -DB2G DSNUBAIC - INCREMENTAL IMAGE COPY DISALLOWED FOR TABLESPACE U9G01T11.TSPSUPP2 DSNUM
1
                FULL IMAGE COPY WILL BE TAKEN
DSNU400I  DSNUBBID - COPY PROCESSED FOR TABLESPACE U9G01T11.TSPSUPP2 DSNUM 1
                NUMBER OF PAGES=3145
                AVERAGE PERCENT FREE SPACE PER PAGE = 1.07
                PERCENT OF CHANGED PAGES = 0.00
                ELAPSED TIME=00:00:03
DSNU428I  DSNUBBID - DB2 IMAGE COPY SUCCESSFUL FOR TABLESPACE U9G01T11.TSPSUPP2 DSNUM 1
DSNU010I  DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=4

```

---



When a recoverable point that indicates that a DFSMS Concurrent COPY is found in SYSCOPY, the RECOVER utility invokes a DFSMSdss RESTORE command to restore the copy, as shown in Example 9-13.

*Example 9-13 Example of RECOVER using a Concurrent COPY*

---

```

DSNU000I  DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = DSNTEX
DSNU050I  DSNUGUTC - RECOVER TABLESPACE U9G01T11.TSPSUPP2 DSNUM 1 REUSE TOCOPY
PAOL0R4.UTIL.TSPSUPP2.T162257
DSNU515I  DSNUCBAL - THE IMAGE COPY DATA SET PAOL0R4.UTIL.TSPSUPP2.T162257 WITH DATE=20090626 AND
TIME=122304
                IS PARTICIPATING IN RECOVERY OF TABLESPACE U7G01T11.TSPSUPP2 DSNUM 1
DSNU421I  DSNUCBUI - START OF DFSMS MESSAGES

PAGE 0001   5695-DF175 DFSMSDSS V2R10.0 DATA SET SERVICES   2009.293 16:34
ADR030I (SCH)-PRIME(01), DCB VALUES HAVE BEEN MODIFIED FOR SYSPRINT
RESTORE DATASET(INCL(PAOL0R1.DSNDBC.U9G01T11.TSPSUPP2.J0001.A001)) -
INDD(SYS00001) CANCE DYNALLOC REPLACE SHARE TOL(ENQF) WAIT(0,0) OUTDY(+
(SBOX58), (SBOX59), (SBOX60), (SBOX57))
ADR101I (R/I)-RI01 (01), TASKID 001 HAS BEEN ASSIGNED TO COMMAND 'RESTORE '
ADR109I (R/I)-RI01 (01), 2009.293 16:34:17 INITIAL SCAN OF USER CONTROL STATEMENTS COMPLETED.
ADR050I (001)-PRIME(01), DFSMSDSS INVOKED VIA APPLICATION INTERFACE
ADR016I (001)-PRIME(01), RACF LOGGING OPTION IN EFFECT FOR THIS TASK
ADR006I (001)-STEND(01), 2009.293 16:34:17 EXECUTION BEGINS
ADR780I (001)-TDDS (01), THE INPUT DUMP DATA SET BEING PROCESSED IS IN LOGICAL DATA SET FORMAT AND
WAS CREATED BY
                DFSMSDSS VERSION 2 RELEASE 10 MODIFICATION LEVEL 0
ADR442I (001)-FRLB0(01), DATA SET PAOL0R1.DSNDBC.U9G01T11.TSPSUPP2.J0001.A001 PREALLOCATED, IN
CATALOG UCAT.VSBOX01,
                ON VOLUME(S): SBOX58
ADR489I (001)-TDLOG(02), CLUSTER PAOL0R1.DSNDBC.U9G01T11.TSPSUPP2.J0001.A001 WAS RESTORED
                CATALOG UCAT.VSBOX01
COMPONENT PAOL0R1.DSNDBD.U9G01T11.TSPSUPP2.J0001.A001
ADR454I (001)-TDLOG(01), THE FOLLOWING DATA SETS WERE SUCCESSFULLY PROCESSED
                PAOL0R1.DSNDBC.U7G01T11.TSPSUPP2.J0001.A001
ADR006I (001)-STEND(02), 2009.293 16:34:21 EXECUTION ENDS
ADR013I (001)-CLTSK(01), 2009.293 16:34:21 TASK COMPLETED WITH RETURN CODE 0000
ADR012I (SCH)-DSSU (01), 2009.293 16:34:21 DFSMSDSS PROCESSING COMPLETE. HIGHEST RETURN CODE IS 0000

```

---

With SHRLEVEL REFERENCE, the objects are not available for update until the copy operation is *logically* completed:

- ▶ All writes are drained, and the objects being copied have a restrictive state of UTRO.
- ▶ The objects are quiesced to ensure that all updated buffers are written to disk before DFSMS Concurrent COPY is invoked. The SYSCOPY records with ICTYPE=Q are inserted to indicate that the objects are successfully quiesced.
- ▶ As soon as the DFSMS Concurrent COPY is logically complete, the objects are drained and the restrictive state is changed from UTRO to UTRW.

When you use the DFSMS Concurrent COPY, you improve the availability of the application data. However, if your table space data sets have more empty space, the physical copy operation may take longer than the DB2 image copy. DFSMS Concurrent COPY does not process the space page map the way DB2 image copy processes it, so the output data sets of DFSMS Concurrent COPY may require more disk space than the output data sets created by the DB2 COPY utility.

### 9.3.1 Concurrent COPY SHRLEVEL REFERENCE using FILTERDDN

Concurrent COPY was intended to work when a list of table spaces is processed in one COPY statement with SHRLEVEL REFERENCE, with each output data set going to the same volume. As each table space is logically completed by DFSMS/DSS, it is put back into UTRW status by DB2, so that other activity can be performed on that table space. Normally, the Concurrent COPY is a two step process: logical completion, and then physical completion.

Ideally, the logical completion of all the table spaces are done together, and all the actual copying can be done later. However, if you specify a list of table spaces to be copied by Concurrent COPY with SHRLEVEL REFERENCE, with each of the copies going to the same output device, all table spaces are put into UTRO and they each remain in UTRO until their respective copies are logically completed, one by one. This means that each table space on the list is not logically completed until the previous one is both logically and physically completed.

This is a big impact on customers who want to cut down the amount of time that their table spaces are unavailable. This is only a problem when each copy output data set is going to the same volume.

To get around this restriction with the existing COPY and DFSMS/DSS infrastructure, you can use the FILTERDDN keyword and copy a list of table spaces using Concurrent COPY to a single image copy output data set (or up to four identical data sets in the case of Local Backup, Recovery site Primary, and Recovery site Backup data sets) instead of the current method of specifying one image copy data set (or up to four in the case of LB, RP, and RB data sets) for each table space in the list. This way, all of the table spaces are processed with a single DFDSS DUMP statement.

Users who do not need relief from this availability problem can use the syntax without the FILTERDDN keyword.

To invoke this functionality, changes to the syntax of existing Concurrent COPY jobs are necessary.

Example 9-14 shows the syntax of a COPY statement without the FILTERDDN.

*Example 9-14 Copy statement without FILTERDDN*

---

```
COPY  TABLESPACE DB1.TS1
      COPYDDN (SYSCOPY1)
      TABLESPACE DB2.TS2
      COPYDDN (SYSCOPY2)
      CONCURRENT
```

---

Example 9-15 shows the FILTERDDN option being used.

*Example 9-15 Copy statement using FILTERDDN*

---

```
COPY  TABLESPACE DB1.TS1
      TABLESPACE DB2.TS2
      FILTERDDN(filterdd)
      COPYDDN (SYSCOPY)
      CONCURRENT
```

---

The usage of the FILTERDDN keyword allows for only one COPYDDN parameter for the entire list of table spaces. Additionally, the use of a new parameter, FILTERDDN, is required. FILTERDDN points to a DD card for a data set that is passed to DFSMS/DSS that contains a list of all the table spaces to be processed. The contents of the data set will be written by DB2, and will be transparent to the user. The user only has to provide a DD card for the FILTERDDN and make sure that the image copy data set is large enough to contain all the table spaces in the list.

When you use the FILTERDDN option, one row will be inserted in SYSIBM.SYSCOPY for each table space or partition (if DSNUM is specified) in the table space list. This is identical to when you are not using the FILTERDDN option. However, the DSNAME column in SYSCOPY will have the same data set name for each table space that is copied at one time using FILTERDDN.

The FILTERDDN keyword is supported when using the DSNUTILS stored procedure to start DB2 utilities.

A sample JCL using FILTERDDN and TEMPLATEs is shown in Example 9-16.

*Example 9-16 Complete JCL using FILTERDDN*

---

```
//PAOLOR41 JOB ,CQM,CLASS=A,MSGCLASS=X,
// NOTIFY=&SYSUID
//JCLL JCLLIB ORDER=DB2V910G.PROCLIB
//UTIL EXEC DSNUPROC,SYSTEM=DB9A,UID='COPY',UTPROC=''
//*
//SYSIN DD *
LISTDEF LISTA1
        INCLUDE TABLESPACE U9G01T11.TSPSUPP2 PARTLEVEL(1)
        INCLUDE TABLESPACE U9G01T11.TSPSUPP PARTLEVEL(1)

TEMPLATE CCDSN UNIT SYSDA
        DSN(PAOLOR4.&STEPNAME..&SN..P&PA..T&TIME.)
        DISP(MOD,CATLG,CATLG)
TEMPLATE FILTDSN UNIT SYSDA
        DSN(PAOLOR4.&STEPNAME..FILTERDD)
        DISP(MOD,CATLG,DELETE)

COPY LIST LISTA1 COPYDDN(CCDSN) FILTERDDN(FILTDSN) CONCURRENT
```

---

The sample job output can be found in Example 9-17. As you can see in the job output, there is only one call to DFSMSdss for both partitions that are being copied. There is only one set of 'ADR006I execution begins - execution ends' messages, as well as message ADR801I, indicating that two data sets were selected when using FILTERDDN, instead of one for every call to DFSMSdss, if FILTERDDN is not used.

*Example 9-17 Job output using FILTERDDN*

---

```

DSNU000I  DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = COPY
DSNU050I  DSNUGUTC - OPTIONS KEY ACAZLHFZ
DSNU1035I DSNUIldr - OPTIONS STATEMENT PROCESSED SUCCESSFULLY
DSNU050I  DSNUGUTC - LISTDEF LISTA1 INCLUDE TABLESPACE U9G01T11.TSPSUPP2 PARTLEVEL(1)
                                INCLUDE TABLESPACE U9G01T11.TSPSUPP PARTLEVEL(1)
DSNU1035I DSNUIldr - LISTDEF STATEMENT PROCESSED SUCCESSFULLY
DSNU050I  DSNUGUTC - TEMPLATE CCDSN UNIT SYSDA
DSN(PAOLOR4.&STEPNAME..&SN..P&PA..T&TIME.) DISP(MOD,CATLG,CATLG)
DSNU1035I DSNUIJTD - TEMPLATE STATEMENT PROCESSED SUCCESSFULLY
DSNU050I  DSNUGUTC - TEMPLATE FILTDSN UNIT SYSDA DSN(PAOLOR4.&STEPNAME..FILTERDD)
                                DISP(MOD,CATLG,DELETE)
DSNU1035I DSNUIJTD - TEMPLATE STATEMENT PROCESSED SUCCESSFULLY
DSNU050I  DSNUGUTC - COPY LIST LISTA1 COPYDDN(CCDSN) FILTERDDN(FILTDSN) CONCURRENT
DSNU1038I DSNUGDYN - DATASET ALLOCATED. TEMPLATE=CCDSN
                                DDNAME=SYS00001
                                DSN=PAOLOR4.UTIL.TSPSUPP.P00001.T183028
DSNU1038I DSNUGDYN - DATASET ALLOCATED. TEMPLATE=FILTDSN
                                DDNAME=SYS00002
                                DSN=PAOLOR4.UTIL.FILTERDD
DSNU421I  DSNUBBCM - START OF DFSMS MESSAGES
PAGE 0001 5695-DF175 DFSMSDSS V2R10.0 DATA SET SERVICES 2009.293 14:30
ADR030I (SCH)-PRIME(01), DCB VALUES HAVE BEEN MODIFIED FOR SYSPRINT
PARALLEL
ADR101I (R/I)-RI01 (01), TASKID 001 HAS BEEN ASSIGNED TO COMMAND 'PARALLEL'
DUM OPT(3) DATAS(FILT(SYS00002)) -
CAN CONC SHA TOL(ENQF) WAIT(0,0) -
OUTDD(SYS00001)
ADR101I (R/I)-RI01 (01), TASKID 002 HAS BEEN ASSIGNED TO COMMAND 'DUM'
ADR109I (R/I)-RI01 (01), 2009.293 14:30:33 INITIAL SCAN OF USER CONTROL STATEMENTS
COMPLETED.
ADR014I (SCH)-DSSU (02), 2009.293 14:30:33 ALL PREVIOUSLY SCHEDULED TASKS COMPLETED.
PARALLEL MODE NOW IN EFFECT
ADR050I (002)-PRIME(01), DFSMSDSS INVOKED VIA APPLICATION INTERFACE
ADR016I (002)-PRIME(01), RACF LOGGING OPTION IN EFFECT FOR THIS TASK
ADR006I (002)-STEND(01), 2009.293 14:30:33 EXECUTION BEGINS
ADR411W (002)-DTDSC(04), DATA SET PAOLOR1.DSNDBC.U9G01T11.TSPSUPP2.J0001.A001 IN CATALOG
UCAT.VSBOX01 ON VOLUME SBOX58
                                WAS NOT SERIALIZED ON REQUEST
ADR411W (002)-DTDSC(04), DATA SET DB2V910G.DSNDBC.U7G01T11.TSPSUPP.I0001.A001 IN CATALOG
UCAT.VSBOX09 ON VOLUME SBOX52
                                WAS NOT SERIALIZED ON REQUEST
ADR801I (002)-DTDSC(01), DATA SET FILTERING IS COMPLETE. 2 OF 2 DATA SETS WERE SELECTED: 0
FAILED SERIALIZATION AND 0
                                FAILED FOR OTHER REASONS.
ADR734I (002)-DTDSC(01), 2009.293 14:30:34 CONCURRENT COPY INITIALIZATION SUCCESSFUL FOR 2
OF 2 SELECTED DATA SETS.
                                SERIALIZATION FOR THIS DATA IS RELEASED IF DFSMSDSS HELD IT. THE
INTERMEDIATE RETURN CODE IS
                                0004.
ADR454I (002)-DTDSC(01), THE FOLLOWING DATA SETS WERE SUCCESSFULLY PROCESSED
                                CLUSTER NAME PAOLOR1.DSNDBC.U9G01T11.TSPSUPP2.J0001.A001
                                CATALOG NAME UCAT.VSBOX01

```

```

      COMPONENT NAME PAOLOR1.DSNDBD.U9G01T11.TSPSUPP2.J0001.A001
      CLUSTER NAME  DB2V710G.DSNDBC.U9G01T11.TSPSUPP.I0001.A001
      CATALOG NAME   UCAT.VSBOX09
      COMPONENT NAME DB2V910G.DSNDBD.U9G01T11.TSPSUPP.I0001.A001
ADR006I (002)-STEND(02), 2009.293 14:30:41 EXECUTION ENDS
ADR013I (002)-CLTSK(01), 2009.293 14:30:41 TASK COMPLETED WITH RETURN CODE 0004
ADR012I (SCH)-DSSU (01), 2009.293 14:30:41 DFSMSDSS PROCESSING COMPLETE. HIGHEST RETURN
CODE IS 0004 FROM:
      TASK      002

```

```

DSNU422I  DSNUBBCM - END OF DFSMS MESSAGE
DSNU413I  -DB2G DSNUBARI - CONCURRENT COPY SUCCESSFUL FOR TABLESPACE U9G01T11.TSPSUPP2
      DSNUM 1
DSNU413I  -DB2G DSNUBARI - CONCURRENT COPY SUCCESSFUL FOR TABLESPACE U9G01T11.TSPSUPP
      DSNUM 1
DSNU401I  DSNUBBCR - CONCURRENT COPY COMPLETE, ELAPSED TIME=00:00:08
DSNU010I  DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0

```

---

When you use a data set that was produced with the FILTERDDN option (and contains the copy data of multiple data sets) in a RECOVER operation, DFDSS will still ask for the data set multiple times instead of restoring all table spaces in a single RESTORE operation.

## 9.4 DSN1COPY

DSN1COPY has two options:

### ► LOB

When using option LOB, DSN1COPY attempts to determine if the input data set is a LOB data set, and if the input data set is a LOB and the option LOB is omitted, DSN1COPY issues an error message and terminates.

Example 9-18 the message that is issued by DSN1COPY.

*Example 9-18 DSN1COPY output*

---

```

//COPY1 EXEC PGM=DSN1COPY,PARM='PAGESIZE(4K),LOB,RESET'
//STEPLIB DD DSN=DB9A9.SDSNEXIT,DISP=SHR
//      DD DSN=DB9A9.SDSNLOAD,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSUT1 DD DSN=DB1.TS1.FD01.T163005,DISP=SHR
//SYSUT2 DD DSN=DB9AU.DSNDBD.DB1.TS1.I0001.A001,DISP=SHR

DSN1999I START OF DSN1COPY FOR JOB DB2R7002 COPY1
DSN1954I DSN1COPY PARAMETER PAGESIZE OR LOB IS MISSING OR INCORRECTLY SPECIFIED
DSN1993I DSN1COPY TERMINATED, 00000000 PAGES PROCESSED

```

---

### ► RESET

Another improvement that comes with DB2 9 is the parameter RESET. If you are copying a compressed table space from a non-data sharing DB2 to another non-data sharing DB2, the dictionary is reset. Until DB2 V8, you had to use a REORG utility with KEEPDICTIONARY=NO.

## 9.4.1 Copying tables from one DB2 subsystem to another

You can copy tables from one subsystem to another. You must ensure that the version information on the target subsystem matches the version information on the source subsystem, or you will need to use the REPAIR utility. One problem that could happen, and you have to take care of, is when you execute an ALTER table on source subsystem, you have to execute the same ALTER on target subsystem before you use DSN1COPY, or unpredictable results can occur. This change should be in the same order and in the same number of units of work.

In Example 9-19, you see an error that could be displayed by DSN1COPY if the ALTER does not occur in the same sequence.

---

### *Example 9-19 Example of problems with DSN1COPY*

---

```
On test table space ( Target )
In ONE unit of work :
Column C1 is altered from CHAR(1) to CHAR(3)
Column C2 is altered from CHAR(2) to CHAR(3)
Columns C22,C23,C24,C25 are added followed by commit
This operations brings the version from 0 to 1.
Run REORG on table space.
On Second unit of work
Column C3 is altered from CHAR(1) to CHAR(2) followed by commit.
This operational brings the version from 1 to 2.
Run REORG on table space.
On production table space (Source)
In ONE unit of work:
Column C1 is altered from CHAR(1) to CHAR(2)
Column C2 is altered from CHAR(2) to CHAR(3)
Column C3 is altered from CHAR(1) to CHAR(2).
Columns C22, C23, C24, C25 are added followed by commit.
This operation brings the version from 0 to 1
Run Reorg on table space.
Take full image copy of production source table space.
DSN1COPY this image copy to the target table space with parameters
CHECK,OBIDXLAT,RESET,FULLCOPY.
Run REPAIR VERSIONS TABLESPACE on target table space.
Run REBUILD INDEX(ALL) on target table space.
Updates to data in target table result in the abend04E.
```

---

In this case, even though the final table column definitions match, abends occur because the version number from the source table does not match the version number in target table definition (OBDREC) due to the different order of ALTER on column C3, which leads to an incorrect code path for update.

The problem is not a defect, but falls under the DSN1COPY restrictions described in *DB2 Version 9.1 Utility Guide and Reference*, SC18-9855, which states:

“If the intended use of DSN1COPY is to move or restore data, ensure that definitions for the source and target table spaces, tables, and indexes are identical. Otherwise, unpredictable results can occur.”

You can solve this problem by deleting the table, rerunning DDL, recovering from the image copy on the source subsystem, deleting and rerunning a DDL on the target subsystem, and then running DSN1COPY, because both are now in Version 1. You probably need to rebind plans on the source subsystem. Alternatively, you should execute a REORG on the source system before the DSN1COPY is taken, and, after executing the DSN1COPY on the data set on the target system, REPAIR VERSIONS should be run.

**Note:** Do not use DSN1COPY to copy XML table spaces from one system to another system, because data dependent information is kept in catalog table XMLSTRINGS and that information is not copied across to the target.







## Recovering data

RECOVER is the primary online utility that is used to recover DB2 objects, table spaces, index spaces, and indexes when a DB2 object is in an unstable or unacceptable state. The restrictive state of the object may be caused by hardware or software failures. The utility recovers data to the current state or to a previous point in time by restoring a copy and then applying log records.

In this chapter, we discuss these RECOVER utility aspects:

- ▶ Recovering data
- ▶ System level point-in-time recovery
- ▶ Recovering to a point in time with consistency
- ▶ RECOVER using the LISTDEF command
- ▶ RECOVER of NOT LOGGED table spaces
- ▶ Recovering a table space that contains LOB or XML data
- ▶ RECOVER CLONE
- ▶ Parallel RECOVER
- ▶ Recovering a data set or partition using DSNUM
- ▶ Performance improvements for RECOVER with concurrent copies
- ▶ LOGONLY
- ▶ LOGAPPLY considerations
- ▶ Fast Log Apply
- ▶ Avoiding specific image copy data sets
- ▶ Restarting RECOVER
- ▶ Displaying progress of RECOVER
- ▶ Index recovery from COPY
- ▶ PIT to a different DB2 function mode
- ▶ General performance recommendations for RECOVER
- ▶ MODIFY RECOVERY
- ▶ REPORT RECOVERY
- ▶ DB2 Recovery Expert
- ▶ DB2 Change Accumulation Tool
- ▶ QUIESCE

## 10.1 Recovering from backup

The RECOVER utility has been enhanced in the following areas:

- ▶ Point-in-time recovery with transactional consistency
- ▶ Displaying the progress of RECOVER
- ▶ Using RECOVER with an earlier copy
- ▶ Volume-based utility enhancements
- ▶ Support Large Block Interface for tapes, which supports up to a 40% reduction in elapsed time for the COPY and RESTORE phase of RECOVER
- ▶ Large format data sets
  - In new-function mode (NFM)
  - Lifts the restriction of a 65,535 track limit per DASD volume
  - Input data sets with transparent support
  - Output data sets using DSN TYPE=LARGE on a DD card or SMS data class
- ▶ MODIFY recovery

## 10.2 System level point-in-time recovery

Enhancements to system level point-in-time recovery for DB2 provide improved usability, more flexibility, and faster backup and recovery. You can recover your data to any point in time, regardless of whether you have uncommitted units of work. Data recovery time improves significantly for large DB2 systems that contain many thousands of objects. Two new utilities are used for system level point-in-time recovery:

- ▶ The BACKUP SYSTEM utility provides fast, nondisruptive, and fuzzy volume-level copies of DB2 databases and logs for an entire DB2 subsystem or DB2 data sharing group. It relies on new DFSMSHsm services in z/OS Version 1 Release 5 that allow for fast volume level backups. BACKUP SYSTEM is less disruptive than using the SET LOG SUSPEND command for copy procedures. An advantage for data sharing environments is that BACKUP SYSTEM has a group scope (compared to SET LOG SUSPEND, which has a member scope).
- ▶ The RESTORE SYSTEM utility recovers a complete DB2 system or a data sharing group to an arbitrary point in time. RESTORE SYSTEM automatically handles any creates, drops, and LOG NO events that may have occurred between the point the backup is taken and the point in time to which you recover.

BACKUP and RESTORE SYSTEM utilities were added in DB2 V8 and use disk volume FlashCopy backups and Copy Pool z/OS DFSMSHsm V1R5 constructs.

The following options have been introduced for RECOVERY:

- ▶ FROMDUMP: Specifies that only dumps of the database Copy Pool are used for the restoration of the data sets. DUMPCLASS (dcl) Indicates the DFSMSHsm dump class that you use to restore the data sets. The FROMDUMP and DUMPCLASS options that you specify for the RECOVER utility override the RESTORE/RECOVER FROM DUMP and DUMPCLASS NAME installation options that you specify in installation panel DSN TIP6.

- ▶ **RESTOREBEFORE X'byte-string'**: Specifies that RECOVER should search for an image copy, concurrent copy, or system-level backup (if yes has been specified for SYSTEM-LEVEL BACKUPS in installation panel DSNTIP6) with an RBA or LRSN value earlier than the specified X'byte-string' value to use in the RESTORE phase.

To avoid specific image copies, concurrent copies, or system-level backups with matching or more recent RBA or LRSN values in START\_RBA, the RECOVER utility applies the log records and restores the object to its current state or the specified TORBA or TOLOGPOINT value. The RESTOREBEFORE value is compared with the RBA or LRSN value in the START\_RBA column in the SYSIBM.SYSCOPY record for those copies. For system-level backups, the RESTOREBEFORE value is compared to the data complete LRSN. If you specify a TORBA or TOLOGPOINT value with the RESTOREBEFORE option, the RBA or LRSN value for RESTOREBEFORE must be lower than the specified TORBA OR TOLOGPOINT value. If you specify RESTOREBEFORE, you cannot specify TOCOPY, TOLASTCOPY, or TOLASTFULLCOPY.

In DB2 9, these utilities are enhanced to use new functions available with z/OS V1R8 DFSMSshm. These enhancements are:

- ▶ The individual table spaces or index spaces can be recovered.
- ▶ The backup can be sent directly to tape.
- ▶ incremental FlashCopy can be used.

For more information, refer to Chapter 13, “BACKUP and RESTORE SYSTEM utilities” on page 339.

## 10.2.1 Using a system-level backup

The RECOVER utility can use a system-level backup of the database Copy Pool as a recovery base. To use a system-level backup, specify YES for the SYSTEM\_LEVEL\_BACKUPS installation option in installation panel DSNTIP6. The RECOVER utility chooses the most recent backup (an image copy, a concurrent copy, or a system-level backup) to restore based on the recovery point for the table spaces or indexes (with the COPY YES attribute) being recovered.

## 10.3 Recovering to a point in time with consistency

Recovering to a point in time is a viable alternative in many situations in which recovery to the current point in time is not possible or not desirable. To make recovery to a point in time work best without losing data consistency, we used to recommend that you take periodic quiesce points at points of consistency.

In a transaction system that has real high volume, taking quiesce points blocks the application access on the objects that are being quiesced. Taking quiesce points frequently may impact the efficiency of a production system. Also, in reality, many point-in-time recoveries must be done using unplanned recovery points. If recovery to an inconsistent point in time must be done, then manual repair of the data left in an inconsistent state is required due to the uncommitted change at recovery point.

This process can be time consuming and often error prone, and requires deeper DB2 knowledge. Also, the ability to recover a list of objects to a point in time with consistency completes the backup and recovery picture when using system-level backups as a recovery base.

Because system-level backups are not taken at consistent points, the enhancement added by this line item allows you to use system-level backups as a recovery base for point-in-time recoveries on a list of objects with consistency. If you want to use system-level backups as a recovery base, we recommend that you set the interval between checkpoints, using the number of minutes instead of the number of log records written.

This enhancement of the RECOVER utility automatically detects the uncommitted transactions that are running at the recover point in time and rolls back their changes on the recovered objects. After recovery, objects are left in their transactionally consistent state. The function is only available in DB2 9 NFM mode.

RECOVER TOLOGPOINT and TORBA have recover with consistency as its default behavior. RECOVER TOCOPY, TOLASTCOPY, and TOLASTFULLCOPY using SHRLEVEL CHANGE copy still have today's recovery behavior, so no consistency is ensured. To achieve consistency with a recovery to a SHRLEVEL CHANGE copy, you can use a desired SHRLEVEL CHANGE copy as the recovery base and specify a recovery point that was right after the copy completed via the TORBA or TOLOGPOINT syntax. You can locate the entry in the SYSCOPY table for the SHRLEVEL CHANGE copy that is being used and use the RBA or LRSN value in the PIT\_RBA column as the recovery point. Use the REPORT RECOVERY utility to see the report of the SYSCOPY records

As part of the recovery process to recover to a point in time with consistency, the changes made on the recovered objects by units of recovery (URs) that are INFLIGHT, INABORT, and POSTPONED ABORT during the recovery time are rolled back. INDOUBT URs during the recovery point are treated as INABORT, and their changes on the recovered objects are also rolled back. INCOMMIT URs are treated as committed, and no rollback happens. If a UR that was active during the recovery to a point in time changed multiple objects in its life span, only the changes made by it on the objects being recovered in current RECOVER utility are rolled back. The changes made by it on other objects are not rolled back.

Therefore, it is essential to include all of the relevant objects in the same RECOVER utility job to ensure consistency from an application point of view. For point-in-time recovery, the RECOVER utility checks to see if all objects in a referential integrity (RI) set, base, and LOB relationship, and base and XML relationship, are recovered in the same list. If all objects are not specified in the list to be recovered, RECOVER sets the appropriate prohibitive pending state. For example, if the parent table space in an RI set is recovered to a point in time, but its dependent table space is not recovered in the same list, then the RECOVER utility places the dependent table space in a check pending (CHKP) state.

During recovery to a point in time with consistency, after the RECOVER utility applies the redo log records in the LOGAPPLY phase, if any DB2 member has an outstanding UR during the recovery point and has modified the objects that are being recovered, the RECOVER utility enters the log analysis phase.

LOGCSR is a new recovery phase and occurs after the RECOVER LOGAPPLY phase. LOGCSR stands for "log analysis current status rebuild". If the RECOVER utility determines that there are no active units of recovery during the recovery point, it skips the LOGCSR phase. During the LOGCSR phase, for each DB2 member that has URs that have changed the objects that are being recovered and were outstanding during the recovery point, the RECOVER utility reads the log forward from the last checkpoint on this DB2 member prior to the recovery point. It identifies the URs that were both active (INFLIGHT, INABORT, INDOUBT, or POSTPONED ABORT) during the recovery point and changes the objects that are being recovered. This is done both sequentially and on a member basis.

Before the start of log analysis for each DB2 member, a new message DSNU1550I is issued. It shows the name of the DB2 member whose log will be analyzed, along with the prior checkpoint RBA value from which the log analysis will start. After the log analysis is done for this DB2 member, a new message DSNU1551I is issued. It marks the end of log analysis for this member and indicates the elapsed time spent on this DB2 member's log analysis. The RECOVER utility then starts the log analysis for the next DB2 member.

After the log analysis is done for all DB2 members, the RECOVER utility issues another new message, DSNU1552I. This message marks the end of the LOGCSR phase and indicates the total elapsed time used by the RECOVER utility for all DB2 members during the LOGCSR phase. The new messages DSNU1550I, DSNU1551I, and DSNU1552I are all part of the RECOVER utility job output.

After completing log analysis in the LOGCSR phase, the RECOVER utility issues a new DSNU1553I message that shows the UR status in a table format. This table shows a combination of the active URs and the recovered objects they changed. It also shows the name of DB2 member that the URs belong to and the RBA value of the earliest log record written by each active UR when it made the change on the object. Each UR's status during the recovery point is also displayed, such as INFLIGHT, INABORT, POSTPONED ABORT, and INDOUBT. INCOMMIT URs are not displayed, because they do not need to be rolled back. This message is displayed in the RECOVER utility job output. This message also shows the total number of INFLIGHT, INABORT, POSTPONED ABORT, and INDOUBT URs during the recovery point. If there is no active UR during the recovery point, these numbers are all zero.

**Note:** If there is a long-running UR, then the LOGUNDO phase could take a significant amount of time. By monitoring the DSNU1553I message, the user may determine that it is more appropriate to terminate (-TERM UTIL command) the recovery and select an alternative recovery point.

If a UR's change to recovered objects needs to be backed out, a new phase called LOGUNDO for the RECOVER utility happens next. The LOGUNDO phase is entered only when the RECOVER job is at a point in time with consistency, or there is a UR that is active during both the recovery point and the changing objects that are being recovered. During the LOGUNDO phase, the RECOVER utility backs out the changes made on recovered objects by active URs on a member basis.

In a data sharing environment, if multiple DB2 members must be processed, a backout is done one member at a time. The sequence of the backout being done among the members is chosen randomly. A fast log apply process is not used during the LOGUNDO phase, even if it is enabled on the DB2 subsystem.

At the beginning of the backout process of each DB2 member, the RECOVER utility issues a new message, DSNU1554I, in the RECOVER job's output. It marks the start of the backout process on this member. The member name is part of the message. During the backout process, the RECOVER utility scans the DB2 log written by the DB2 member that is being processed backwards in one pass. The backward scan starts from the recovery point. During the backward process, changes made on the recovered objects by this DB2 member's active URs are undone. The active URs were identified in the LOGCSR phase. The backward log scan continues until all the changes made on the recovered objects by this DB2 member's active URs are backed out.

During the backout phase, the RECOVER utility periodically issues another new message, DSNU1555I, into its job output after it processes a certain number of log records. This message shows the RBA value of the current log record that is being processed by the RECOVER utility and the RBA value of the last log record that is read in this member's LOGUNDO phase. By monitoring this message, the user knows the progress of the RECOVER LOGUNDO phase on this DB2 member. In extreme cases, for example, on one DB2 member, a long running UR may be changing objects that were recovered a long time ago and were active at the recovery point. The backout process for this member could take a long time since it has to roll back all the changes made by this UR on the recovered objects. This case should not happen often if all the applications commit on a regular basis.

If this case does happen, you can terminate the current RECOVER job and perform another RECOVER job on the same objects by choosing a different recover to point in time where the UR with the problem is not active at that time. The -TERM UTIL command can terminate the RECOVER utility during the LOGAPPLY, LOGCSR, and LOGUNDO phases. The command will be recognized at the next commit point and the utility will terminate itself gracefully. In prior releases, the -TERM UTIL command was ignored in the LOGAPPLY phase.

After the backout process on one DB2 member is done, the RECOVER utility issues another new message, DSNU1556I, to indicate that the backout is done on this member. The member name is included in the message. The elapsed time spent in the UNDO processing for this DB2 member is also shown in this message. This message is also saved in the RECOVER utility job output. If more DB2 members have log records that need to be processed, the RECOVER utility starts the backout process for the next DB2 member. When no more DB2 members are to be processed, the RECOVER utility issues a new message, DSNU1557I, in the RECOVER utility job output. This message marks the end of the LOGUNDO phase and indicates the total elapsed time spent in this phase.

Figure 10-1 shows the Recovery to PIT consistency highlights.

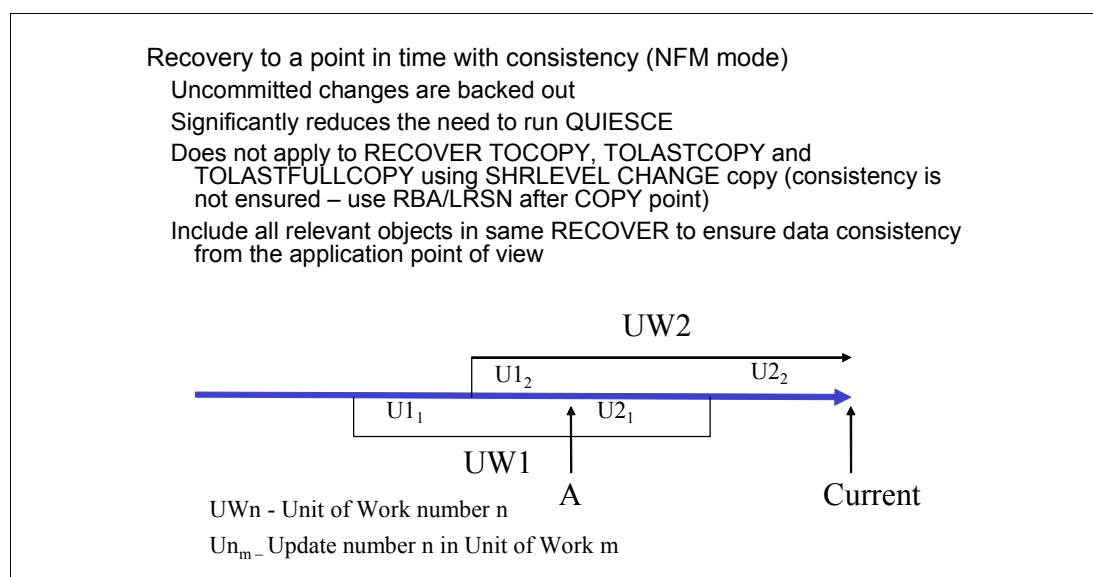


Figure 10-1 Recovery to PIT with consistency highlights

DB2 9 for z/OS takes considerable measures to reduce the time that is required for manual interventions to perform such an operation. The following steps are improved during a RECOVER to a point-in-time process:

1. Automatically detect the uncommitted transactions that are running at the point-in-time recovery point.
2. Roll back changes on the object to be recovered to ensure data consistency after the point-in-time recovery. No fast log apply function is used. Here are the particulars:
  - a. Changes made on the recovered objects by URs that are INFLIGHT, INABORT, POSTPONED ABORT during the recovery time are rolled back.
  - b. URs that are INDOUBT during the recovery point are treated as INABORT, and their changes on the recovered objects are also rolled back.
  - c. INCOMMIT URs are treated as committed, and no rollback occurs.
  - d. If a UR changes multiple objects in its life span:
    - i. Only changes made by it on the objects that are being recovered are rolled back.
    - ii. Changes made by it on other objects are not rolled back.
3. Leave the recovered objects in a consistent state from a transaction point of view.

DB2 objects can now be recovered to any previous point in time with full data integrity. In addition, these improvements allow you to avoid running the QUIESCE utility regularly so you can reduce the disruption to DB2 users and applications.

**Important:** A prerequisite for using this recovery function with system-level backups is DFSMSHsm Version 1.8 or later. If you have enabled system level backups in DSNZPARM and are not at the proper level, you receive the message:

```
DSNU1608I RECOVER SYSTEM UTILITY FAILED. REQUIRED DFSMSHSM SERVICES NOT AVAILABLE
```

To use this feature with native image copies and exclude system-level backups, you must first disable the entry in DSNZPARM:

```
SYSTEM_LEVEL_BACKUPS=NO
```

Remember that in a data sharing environment, you must specify TOLOGPOINT instead of TORBA, or you receive this message:

```
DSNU529I -D2S1 293 11:30:44.94 DSNUCAIN - INVALID SPECIFICATION OF TORBA = X'0071F3F8A000' The RBA is within the range of the used logs of the specific group member
```

Example 10-1 shows the output of a RECOVER with consistency.

**Example 10-1 RECOVER with consistency**

---

```

1DSNU000I   293 12:17:51.10 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = HEINZREC
DSNU1044I   293 12:17:51.15 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
ODSNU050I   293 12:17:51.15 DSNUGUTC - RECOVER TABLESPACE DSN8D81A.DSN8S81S TOLOGPOINT X'BF94F73EE1E6'
DSNU459I    293 12:17:51.28 DSNUCBMD - SYSCOPY P RECORD ENCOUNTERED FOR TABLESPACE DSN8D81A.DSN8S81S , PIT
RBA=BF927F24395F
DSNU515I    293 12:17:51.28 DSNUCBAL - THE IMAGE COPY DATA SET BAEK.HEELA.T153805.REORG.SYSCOPY1 WITH DATE=20061018 AND
TIME=113827
                IS PARTICIPATING IN RECOVERY OF TABLESPACE DSN8D81A.DSN8S81S
DSNU504I    293 12:17:55.07 DSNUCBMD - MERGE STATISTICS FOR TABLESPACE DSN8D81A.DSN8S81S -
                NUMBER OF COPIES=1
                NUMBER OF PAGES MERGED=21134
                ELAPSED TIME=00:00:03
DSNU830I   -D2S1 293 12:17:51.34 DSNUCARS - INDEX DSN8810.XPARTS IS IN REBUILD PENDING
DSNU831I   -D2S1 293 12:17:51.35 DSNUCARS - ALL INDEXES OF DSN8D81A.DSN8S81S ARE IN REBUILD PENDING
DSNU578I   -D2S1 293 12:17:55.11 DSNUCALA - SYSLGRNX INFORMATION FOR MEMBER D2S1
DSNU513I   -D2S1 293 12:17:55.11 DSNUCALA - RECOVER UTILITY LOG APPLY RANGE IS RBA 00716C90DF2C LRSN BF92A6F681E4 TO
                RBA 007175526260 LRSN BF92A978DC4F
DSNU579I   -D2S1 293 12:17:55.29 DSNUCACL - RECOVER UTILITY LOG APPLY AT LOGPOINT BF92A6F70971
DSNU579I   -D2S1 293 12:17:59.58 DSNUCACL - RECOVER UTILITY LOG APPLY AT LOGPOINT BF92A703A364
DSNU579I   -D2S1 293 12:18:34.53 DSNUCACL - RECOVER UTILITY LOG APPLY AT LOGPOINT BF94F73ED03E
DSNU1510I   293 12:18:34.77 DSNUCBLA - LOG APPLY PHASE COMPLETE, ELAPSED TIME = 00:00:39
DSNU1550I   -D2S1 293 12:18:34.78 DSNUCALC - LOGCSR IS STARTED FOR MEMBER D2S1, PRIOR CHECKPOINT RBA = X'0071F0001AE0'
DSNU1551I   -D2S1 293 12:18:35.01 DSNUCALC - LOGCSR IS FINISHED FOR MEMBER D2S1, ELAPSED TIME = 00:00:00
DSNU1552I   -D2S1 293 12:18:35.01 DSNUCALC - LOGCSR PHASE COMPLETE, ELAPSED TIME = 00:00:00
DSNU1553I   -D2S1 293 12:18:35.01 DSNUCALC - RECOVER DETECTS THE FOLLOWING ACTIVE URS:
                INFLIGHT = 1, INABORT = 0, INDOUBT = 0, POSTPONED ABORT = 0
                MEM   T   CONNID   CORRID   AUTHID   PLAN   S   URID   DATE   TIME
                D2S1   B   TSO     RUFLAIR   RUFLAIR   DSNESPCS F 0071E55E52B4 2006-10-20 14.46.11
                DBNAME SPACENAME DBID/PSID PART   RBA
                DSN8D81A DSN8S81S 0105/000A 0000 BF94F73035FD
DSNU1554I   -D2S1 293 12:18:35.15 DSNUCALU - LOGUNDO IS STARTED FOR MEMBER D2S1
DSNU1555I   -D2S1 293 12:18:35.32 DSNUCACL - RECOVER LOGUNDO STATUS: LOG RECORD AT RBA X'BF94F73EC7DA' TO RBA
X'BF94F73035FD' ON MEMBER D2S1
DSNU1555I   -D2S1 293 12:18:58.91 DSNUCACL - RECOVER LOGUNDO STATUS: LOG RECORD AT RBA X'BF94F7304160' TO RBA
X'BF94F73035FD' ON MEMBER D2S1
DSNU1556I   -D2S1 293 12:18:58.99 DSNUCALU - LOGUNDO IS FINISHED FOR MEMBER D2S1, ELAPSED TIME = 00:00:23
DSNU1557I   -D2S1 293 12:18:58.99 DSNUCALU - LOGUNDO PHASE COMPLETE, ELAPSED TIME = 00:00:23
DSNU500I    293 12:18:59.15 DSNUCBDR - RECOVERY COMPLETE, ELAPSED TIME=00:01:08
DSNU010I    293 12:18:59.15 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=4

```

---

If you recover data to a point in time, RECOVER puts any associated index spaces in REBUILD-pending status. If you specify the TOLOGPOINT or TORBA option to recover data to a point in time, RECOVER can put any associated index spaces in REBUILD-pending status only if the indexes are not recovered in the same RECOVER statement as their corresponding table space. You must run REBUILDINDEX to remove the REBUILD-pending status from the index space. If you use the RECOVER utility to recover a point-in-time object that is part of a referentially related table space set, a base table space and LOB table space set, or a base table space and XML table space set, you must ensure that you recover the entire set of table spaces. If you do not include every member of the set, or if you do not recover the entire set to the same point in time, RECOVER sets the Auxiliary CHECK-pending status on for all dependent table spaces, base table spaces, or LOB table spaces in the set.



**Important:**

- RECOVER cannot recover an index has been altered to PADDED or NOT PADDED. Instead, you should rebuild the index.
- If a table space or partition in reordered row format is recovered to a point in time when the table space or partition was in basic row format, then the table space or partition will revert to basic row format. Similarly, if a table space or partition in basic row format is recovered to a point in time when the table space or partition was in reordered row format, then the table space or partition will revert to reordered row format.

Example 10-2 shows the output of a recovery job with a TORBA. In this example, there was a single transaction active at the recovery point. The transaction affected two tables that were part of an RI set (in two table spaces).

**Example 10-2 RECOVER TORBA**


---

```

1DSNU000I 052 19:58:01.49 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = COPYTS
DSNU1044I 052 19:58:01.53 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
ODSNU050I 052 19:58:01.54 DSNUGUTC - LISTDEF ROY INCLUDE TABLESPACE DSN00218.ROYC INCLUDE TABLESPACE
DSNU0217.ROYP
DSNU1035I 052 19:58:01.54 DSNUIldr - LISTDEF STATEMENT PROCESSED SUCCESSFULLY
ODSNU050I 052 19:58:01.54 DSNUGUTC - RECOVER LIST ROY TORBA X'00002C6A42C3'
DSNU1033I 052 19:58:01.55 DSNUGULM - PROCESSING LIST ITEM: TABLESPACE DSN00218.ROYC
DSNU1033I 052 19:58:01.55 DSNUGULM - PROCESSING LIST ITEM: TABLESPACE DSN00217.ROYP
DSNU532I 052 19:58:01.57 DSNUCBMD - RECOVER TABLESPACE DSN00218.ROYC START
DSNU515I 052 19:58:01.57 DSNUCBAL - THE IMAGE COPY DATA SET PAOLOR9.COPYA3 WITH DATE=20070221 AND TIME=195306
IS PARTICIPATING IN RECOVERY OF TABLESPACE DSN00218.ROYC
DSNU504I 052 19:58:01.94 DSNUCBMD - MERGE STATISTICS FOR TABLESPACE DSN00218.ROYC -
NUMBER OF COPIES=1
NUMBER OF PAGES MERGED=3
ELAPSED TIME=00:00:00
DSNU532I 052 19:58:01.96 DSNUCBMD - RECOVER TABLESPACE DSN00217.ROYP START
DSNU515I 052 19:58:01.96 DSNUCBAL - THE IMAGE COPY DATA SET PAOLOR9.COPYP3 WITH DATE=20070221 AND TIME=195235
IS PARTICIPATING IN RECOVERY OF TABLESPACE DSN00217.ROYP
DSNU504I 052 19:58:02.32 DSNUCBMD - MERGE STATISTICS FOR TABLESPACE DSN00217.ROYP -
NUMBER OF COPIES=1
NUMBER OF PAGES MERGED=2
ELAPSED TIME=00:00:00
DSNU830I -DB9B 052 19:58:01.59 DSNUCARS - INDEX PAOLOR9.CIN IS IN REBUILD PENDING
DSNU831I -DB9B 052 19:58:01.59 DSNUCARS - ALL INDEXES OF DSN00218.ROYC ARE IN REBUILD PENDING
DSNU830I -DB9B 052 19:58:01.98 DSNUCARS - INDEX PAOLOR9.ROYPABCD.#_JWY IS IN REBUILD PENDING
DSNU830I -DB9B 052 19:58:01.98 DSNUCARS - INDEX PAOLOR9.PIN IS IN REBUILD PENDING
DSNU831I -DB9B 052 19:58:01.98 DSNUCARS - ALL INDEXES OF DSN00217.ROYP ARE IN REBUILD PENDING

DSNU513I -DB9B 052 19:58:02.34 DSNUCALA - RECOVER UTILITY LOG APPLY RANGE IS RBA 00002C699158 LRSN 00002C699158 TO
RBA 00002C6A42C3 LRSN 00002C6A42C3
DSNU1510I 052 19:58:03.42 DSNUCBLA - LOG APPLY PHASE COMPLETE, ELAPSED TIME = 00:00:01
DSNU1550I -DB9B 052 19:58:03.42 DSNUCALC - LOGCSR IS STARTED FOR MEMBER , PRIOR CHECKPOINT RBA =
X'00002A0192C4'
DSNU1551I -DB9B 052 19:58:04.24 DSNUCALC - LOGCSR IS FINISHED FOR MEMBER , ELAPSED TIME = 00:00:00
DSNU1552I -DB9B 052 19:58:04.24 DSNUCALC - LOGCSR PHASE COMPLETE, ELAPSED TIME = 00:00:00

```

---

The RECOVER is for both table spaces. The transaction is identified as INFLIGHT. Both table spaces are shown as needing to be backed out. See DSNU1553I.

**Important:** When a table space is being recovered, it is essential that all other table spaces that are being changed by the same transactions at the recovery point are included in the *same* RECOVER statement. In Example 10-2, we use a LISTDEF to do this task. RECOVER will never back out changes for objects that are not being recovered.

**Considerations:**

- ▶ You cannot recover an index or index space to a point in time prior to an ALTER INDEX REGENERATE. Rebuild it by using the REBUILD INDEX utility.
- ▶ You cannot recover to a point in time after ALTER TABLE ADD COLUMN and before ALTER COLUMN DROP DEFAULT.
- ▶ You cannot recover a piece of a linear table space to a point in time when the table space was in basic row format and is now in reordered row format, or vice versa. The entire table space must be recovered.
- ▶ You cannot recover a point-in-time RECOVER INDEX if the recovery point precedes the first ALTER INDEX, in Version 8 new-function mode, that created a new index version.

If RECOVER fails in the LOGCSR phase and you restart the utility, the utility restart behavior is RESTART(PHASE).

If RECOVER fails in the LOGUNDO phase and you restart the utility, the utility repeats the RESTORE, LOGAPPLY, LOGCSR, and LOGUNDO phases for only those objects that had active units of recovery that needed to be handled and that did not complete undo processing prior to the failure.

## 10.4 RECOVER using the LISTDEF command

LISTDEF assists you in defining the DB2 objects with wildcard representation. Refer to Chapter 3, “Simplifying utilities with wildcarding and templates” on page 47 for details about LISTDEF and TEMPLATE statements.

The RECOVER utility supports the list generated by the previous LISTDEF command. Example 10-3 shows a sample job with the OPTIONS PREVIEW command that lists the table spaces, which will participate in the RECOVER utility.

*Example 10-3 RECOVER using a LISTDEF command with a wildcard*

---

```
//STEP1    EXEC DSNUPROC,SYSTEM=DB2G,UID=PAOLR1,
//          UTSTATS=' '
//SYSIN    DD *
           OPTIONS PREVIEW
           LISTDEF DB01A
             INCLUDE TABLESPACE U7G01T11.*
             INCLUDE TABLESPACE U7G01T11.TSLINEI PARTLEVEL
             EXCLUDE TABLESPACE U7G01T11.TSLINEI
             EXCLUDE TABLESPACE U7G01T11.TESTNPIS
             EXCLUDE TABLESPACE U7G01T11.TSP*
           RECOVER LIST DB01A PARALLEL 10
```

---

Example 10-4 is the output from the JCL in Example 10-3 on page 260. Note that the LISTDEF expansion does not include the table spaces that are explicitly excluded in the LISTDEF command.

*Example 10-4 LISTDEF OPTIONS PREVIEW job output*

---

```

DSNU000I  DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = PAOLOR1
DSNU050I  DSNUGUTC -  OPTIONS PREVIEW
DSNU1000I  DSNUGUTC - PROCESSING CONTROL STATEMENTS IN PREVIEW MODE
DSNU1035I  DSNUIHDR - OPTIONS STATEMENT PROCESSED SUCCESSFULLY
DSNU050I  DSNUGUTC - LISTDEF DB01A INCLUDE TABLESPACE U7G01T11.* INCLUDE TABLESPACE
U7G01T11.TSLINEI PARTLEVEL
EXCLUDE TABLESPACE U7G01T11.TSLINEI EXCLUDE TABLESPACE U7G01T11.TESTNPIS EXCLUDE TABLESPACE
U7G01T11.TSP*
DSNU1035I  DSNUIHDR - LISTDEF STATEMENT PROCESSED SUCCESSFULLY
DSNU1020I -DB2G DSNUIlsa - EXPANDING LISTDEF DB01A
DSNU1021I -DB2G DSNUIlsa - PROCESSING INCLUDE CLAUSE TABLESPACE U7G01T11.*
DSNU1022I -DB2G DSNUIlsa - CLAUSE IDENTIFIES 11 OBJECTS
DSNU1021I -DB2G DSNUIlsa - PROCESSING INCLUDE CLAUSE TABLESPACE U7G01T11.TSLINEI
DSNU1022I -DB2G DSNUIlsa - CLAUSE IDENTIFIES 3 OBJECTS
DSNU1021I -DB2G DSNUIlsa - PROCESSING EXCLUDE CLAUSE TABLESPACE U7G01T11.TSLINEI
DSNU1022I -DB2G DSNUIlsa - CLAUSE IDENTIFIES 1 OBJECTS
DSNU1021I -DB2G DSNUIlsa - PROCESSING EXCLUDE CLAUSE TABLESPACE U7G01T11.TESTNPIS
DSNU1022I -DB2G DSNUIlsa - CLAUSE IDENTIFIES 1 OBJECTS
DSNU1021I -DB2G DSNUIlsa - PROCESSING EXCLUDE CLAUSE TABLESPACE U7G01T11.TSP*
DSNU1022I -DB2G DSNUIlsa - CLAUSE IDENTIFIES 4 OBJECTS
DSNU1023I -DB2G DSNUIlsa - LISTDEF DB01A CONTAINS 8 OBJECTS
DSNU1010I  DSNUGPVV - LISTDEF DB01A EXPANDS TO THE FOLLOWING OBJECTS:
          LISTDEF DB01A -- 00000008 OBJECTS
          INCLUDE TABLESPACE U7G01T11.TSORDER
          INCLUDE TABLESPACE U7G01T11.TSCUST
          INCLUDE TABLESPACE U7G01T11.TSSUPPLY
          INCLUDE TABLESPACE U7G01T11.TSNATION
          INCLUDE TABLESPACE U7G01T11.TSREGION
          INCLUDE TABLESPACE U7G01T11.TSLINEI PARTLEVEL(00001)
          INCLUDE TABLESPACE U7G01T11.TSLINEI PARTLEVEL(00002)
          INCLUDE TABLESPACE U7G01T11.TSLINEI PARTLEVEL(00003)
DSNU050I  DSNUGUTC - RECOVER LIST DB01A PARALLEL 5
DSNU010I  DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0

```

---

The utility list manager will invoke RECOVER once for the entire list generated by the LISTDEF. Therefore, LIST cannot be used with these commands:

- ▶ PAGE.
- ▶ ERROR RANGE.
- ▶ TOCOPY.
- ▶ DSNUM. Instead use PARTLEVEL, as shown in Example 10-3 on page 260.

**Tip:** Use RECOVER on a LISTDEF list with PARALLEL and fast LOGAPPLY to maximize parallelism. The PARALLEL option prompts the RECOVER utility to sort the objects in the list by the device type of the input image copies (tape or DASD), by size, and also by type of image copy (image copy versus concurrent copy). If the input image copies are stacked on tape, then they are sorted by tape stack (that is, the tape VOLSERS) and ascending file sequence numbers on the tape stack to avoid demounts and rewinds of the tape volumes during the RESTORE phase.

**Note:** You must specify the CLONE keyword if you want process clone data. The use of CLONED YES on the LISTDEF statement is not sufficient.

## 10.5 RECOVER of NOT LOGGED table spaces

You can recover NOT LOGGED table spaces to any recoverable point.

The NOT LOGGED logging attribute of tables spaces does not mean that the contents of a table space are non-recoverable. However, the modifications to a table space that are not logged are non-recoverable. Recovery can be to any recoverable point. A recoverable point is established when a table space is altered from logged to not logged or when an image copy is taken against a table space that is not logged. The creation of the table space is also a recoverable point. Be aware that If you update a table space while it has the NOT LOGGED logging attribute, the table space is marked Informational Copy Pending (ICOPY). You can use the DISPLAY DATABASE ADVISORY command to display the ICOPY status for table spaces. The TORBA or TOLOGPOINT keywords can also be used for a point-in-time recovery on an object that is not logged, but the RBA or LRSN must correspond to a recoverable point or message DSNU1504I is issued.

You can use RECOVER with the TOCOPY, TOLASTFULLCOPY, or TOLASTCOPY keyword to identify which image copy to use. You should run RECOVER with the TOLOGPOINT option to identify a common, recoverable point for all objects in a set of objects. You cannot use RECOVER with the LOGONLY keyword.

If a base table space is altered so that it is not logged, and its associated LOB table spaces already have the NOT LOGGED attribute, then the point where the table space is altered is not a recoverable point. Use image copies that are taken for NOT LOGGED table spaces that have associated LOB or XML table spaces as a recovery set, so that the base table space and all the associated LOB or XML table spaces are copied at the same point in time. This way, a RECOVER TO LASTCOPY of the entire set results in consistent data across the base table space and all of the associated LOB or XML table spaces.

Note that RECOVER TOLASTCOPY cannot operate on a list of objects, so it is best to use RECOVER TOLOGPOINT on the set of objects where the RBA or LRSN corresponds to the syscopy START\_RBA value for all objects.

Specifying REBUILD INDEX SHRLEVEL(CHANGE) is not allowed on a NOT LOGGED table space because there are no log records available (You will get message DSNU1152I).

## 10.6 Recovering a table space that contains LOB or XML data

You can use the RECOVER utility to recover XML objects. You do not have to specify any additional keywords in the RECOVER statement. Nevertheless, when recovering page sets that contain XML columns, it is not enough to just recover the base table space. You must recover your XML table space or table spaces if your table contains more than one column with data type XML, just like you recover related LOB table spaces or related RI table spaces. You can use REPORT TABLESPACESET to identify related table spaces and their index spaces. Additionally, you must recover or rebuild all indexes that are defined on your XML table, including indexes that you have created for performance or unique reasons, but also the NodeID and DocID index. REBUILD INDEX SHRLEVEL CHANGE for XML indexes is not allowed.

The RECOVER utility sets the auxiliary warning status for LOB or XML table spaces if it finds at least one invalid LOB or XML column. DB2 marks a LOB or XML column invalid if all of the following conditions are true:

- ▶ The LOB/ XML table space was defined with the LOG(NO).
- ▶ The LOB/ XML table space was recovered.
- ▶ The LOB/ XML was updated since the last image copy.

The status of LOB/ XML related objects depends on the type of recovery that is performed.

If you recover all of the following objects for all LOB/ XML columns in a single RECOVER to the present point in time, no pending status is ever set, even if all of the objects were not recovered:

- ▶ Base table space
- ▶ Index on the auxiliary table
- ▶ LOB table space
- ▶ XML table space

## 10.7 RECOVER CLONE

For RECOVER, if you specify CLONE, DB2 only recovers clone table data in the specified table space or the specified index spaces that contain indexes on table clones. If you specify REBUILD INDEX CLONE, only those indexes are rebuilt that are on table clones. The keywords CLONE and STATISTICS are mutually exclusive. DB2 does not collect statistics for table clones.

A point-in-time recovery cannot be done to a time that precedes the most recent EXCHANGE statement for an object currently involved in cloning, or one that was previously involved in cloning. The time of the most recent EXCHANGE for a table space can be determined by running REPORT RECOVERY for the object.

Base and clone objects must be recovered in separate RECOVER utility statements.

## 10.8 Parallel RECOVER

DB2 objects can be recovered in parallel using the PARALLEL keyword. The number of parallel objects to recover is determined by the value following the PARALLEL keyword. PARALLEL also specifies the maximum number of objects in the list that are to be restored in parallel from system-level backups that have been dumped to tape. The processing may be limited by DFSMSHsm. RECOVER attempts to retain tape mounts and determines the order in which objects are to be restored. If the *num-objects* value is 0 or is not specified, or you do not specify TAPEUNITS, RECOVER will determine the optimum number of objects to process in parallel based on available storage, CTHREAD, and IDBACK values in DSNZPARM.

The RECOVER utility processes the objects to be recovered and inputs image copies on DASD as follows:

- ▶ The objects are sorted by size.
- ▶ The smallest objects are restored first.
- ▶ As each of the table spaces in the list complete the restoration process, the next object in the sorted list will be processed.

When the input image copies are stacked on tape and, for example, PARALLEL(4) has been specified, but the number of objects to be restored in parallel is limited to two because there are two input tape stacks, then the RECOVER utility will sort the list into smaller lists based on ascending file sequence number (this number is found on the tape stacks).

Assuming the number of optimum processes to be N, Table 10-1 shows the maximum number of parallel tasks for a specified *num-object* value. RECOVER does not perform parallel processing for objects that are in backup or fallback recovery. With the DSNUM parameter, you can specify what partitions will be restored in parallel, as shown in Example 10-7 on page 267.

*Table 10-1 Total number of processes used for RECOVER in parallel*

PARALLEL ( <i>num-objects</i> )	Number of parallel tasks
0 or Not specified	N
> N	N
< N	<i>num-objects</i>

The job in Example 10-3 on page 260 was rerun without the OPTIONS PREVIEW command and PARALLEL 10. The output in Example 10-5 indicates that RECOVER processed seven objects in parallel.

*Example 10-5 Partial job output with seven objects in parallel*

```

DSNU000I  DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = PAOLOR1
DSNU050I  DSNUGUTC - LISTDEF DB01A INCLUDE TABLESPACE U7G01T11.* INCLUDE TABLESPACE
U7G01T11.TSLINEI PARTLEVEL
EXCLUDE TABLESPACE U7G01T11.TSLINEI EXCLUDE TABLESPACE U7G01T11.TESTNPIS EXCLUDE TABLESPACE
U7G01T11.TSP*
DSNU1035I  DSNUIHDR - LISTDEF STATEMENT PROCESSED SUCCESSFULLY
DSNU050I  DSNUGUTC - RECOVER LIST DB01A PARALLEL 10
DSNU397I  DSNUCBMT - NUMBER OF TASKS CONSTRAINED BY CONNECTIONS
DSNU427I  DSNUCBMT - OBJECTS WILL BE PROCESSED IN PARALLEL,
              NUMBER OF OBJECTS = 7

```

**Note:** If the number of parallel tasks are limited, the following message can be issued:

‘DSNU397I DSNUCBMT - NUMBER OF TASKS CONSTRAINED BY CONNECTIONS’

Check the values of IDBACK and CTHREAD in DSNZPARM. Increase the value of IDBACK.

Figure 10-2 shows the schematic diagram of the parallel processes involved in a recovery job with PARALLEL 3. Each object is serviced by two subtasks, one to READ the input data from an external data set, the second to WRITE the data, piped from the READ subtask, to the object space. When recovery of object 3 finishes, the subtask initiates recovery of object 4 while the other subtasks are still processing object 1 and 2, respectively.

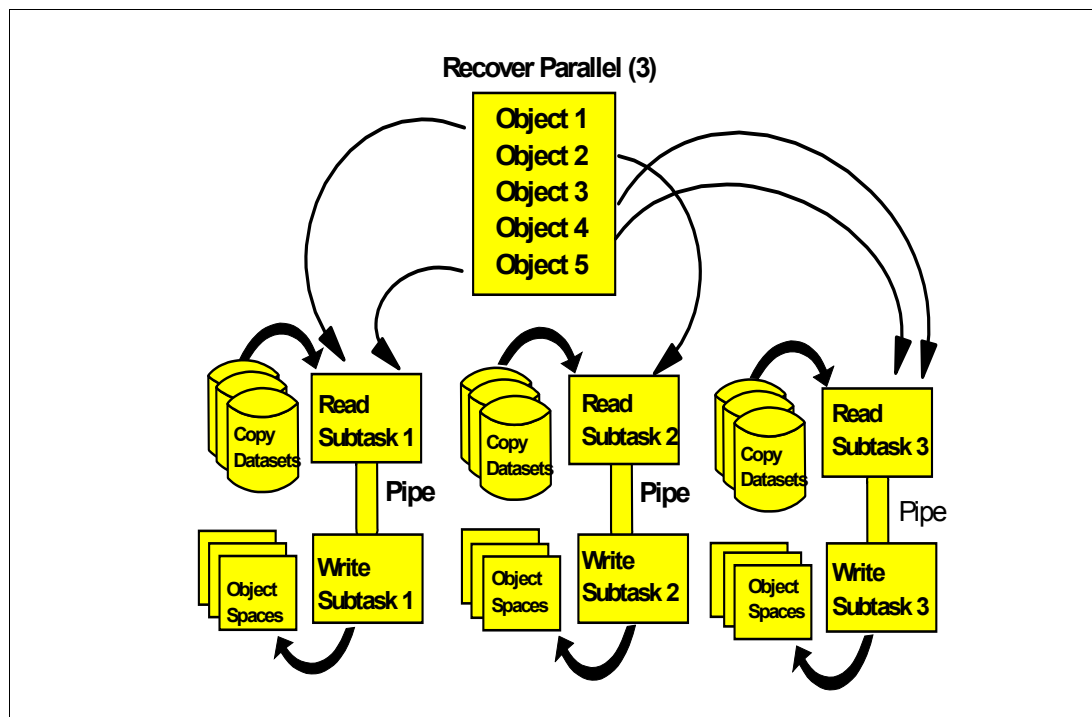


Figure 10-2 RECOVER with PARALLEL keyword

The Log Apply processing does not begin until all objects have been restored. There is parallelism in the LOGAPPLY phase via Fast Log Apply processing.

### 10.8.1 Retaining tape mounts

The RECOVER utility can automatically retain the tape volumes for the input image copies when a list of objects is being recovered. For input image copies (for the objects being recovered) that are stacked on one or more tape volumes, you do not need to code JCL DD statements to retain the tape volumes on the tape drive. Instead, you can use the PARALLEL and TAPEUNITS keywords. The PARALLEL keyword directs the RECOVER utility to process the objects in parallel. The objects will be sorted based on how the input image copies are stacked on tape to maximize efficiency during the RESTORE phase by retaining the tape volumes on the tape drive and by restoring the input image copies in the right order (by ascending file sequence numbers). The TAPEUNITS keyword will limit the number of tape units (or drives) that the RECOVER utility will use during the RESTORE phase. In special cases, RECOVER cannot retain all of the tape volumes, so the tape volumes may be demounted and deallocated even if the PARALLEL and TAPEUNITS keywords are specified. The TAPEUNITS keyword applies only to tape drives that are dynamically allocated. The TAPEUNITS keyword does not apply to JCL-allocated tape drives.

The total number of tape drives that are allocated for the RECOVER job is the sum of the JCL-allocated tape drives, and the number of tape drives, which is determined as follows:

- ▶ The specified value for TAPEUNITS.
- ▶ The value that is determined by the RECOVER utility if you omit the TAPEUNITS keyword. The number of tape drives that RECOVER attempts to allocate is determined by the object in the list that requires the most tape drives.

**Note:** If there is a problem allocating, opening, or reading from an image copy, then the restoration of the object will be deferred until all of the other objects in the list have been restored. This is done to ensure that the rest of the objects can be processed in the most efficient manner.

## 10.8.2 Restrictions of the RECOVER utility with keyword PARALLEL

The following restrictions apply to the RECOVER utility with keyword PARALLEL:

- ▶ DB2 may reduce the number of objects processed in parallel if there is a virtual storage constraint in the utility batch address space.
- ▶ DB2 disables parallelism if the PARALLEL keyword is omitted.
- ▶ The keywords PARALLEL and LOGONLY are mutually exclusive for the RECOVER utility.
- ▶ If you specify PARALLEL, you cannot specify TOCOPY, TOLASTCOPY, or TOLASTFULLCOPY.

In order to benefit from parallelism, the RECOVER utility must be run with the PARALLEL keyword and an *object list* must be provided. RECOVER supports two forms of object lists:

- ▶ The object-list explicitly defined for RECOVER utility, as shown in Example 10-6.
- ▶ The LISTDEF command to build the object-list and pass the list to RECOVER using the LIST keyword, as discussed in 10.4, “RECOVER using the LISTDEF command” on page 260.

*Example 10-6 RECOVER with object-list*

---

```
//STEP1    EXEC DSNUPROC,SYSTEM=DB2G,UID=PAOL0R1,
//          UTSTATS=''
//SYSIN    DD *
RECOVER
    TABLESPACE U7G01T11.TSORDER
    TABLESPACE U7G01T11.TSCUST
    TABLESPACE U7G01T11.TSSUPPLY
    TABLESPACE U7G01T11.TSNATION
    TABLESPACE U7G01T11.TSREGION
    PARALLEL
```

---

In both cases, RECOVER will dispatch the optimal number of subtasks to recover the objects in parallel.

We recommend using the PARALLEL option with the value 0 or without a value, and let RECOVER determine the optimum number of parallel subtasks.



## 10.9 Recovering a data set or partition using DSNUM

You can use the RECOVER utility to recover individual partitions and data sets. The phases for data set recovery are the same as for table space recovery. If image copies are taken at the data set level, RECOVER must be performed at the data set level. To recover the whole table space, you must recover all the data sets individually in one or more RECOVER steps. If recovery is attempted at the table space level, DB2 returns an error message. If image copies are taken at the table space, index, or index space level, you can recover individual data sets by using the DSNUM parameter.

With DSNUM, you identify a partition within a partitioned table space or a partitioned index, or a data set within a non-partitioned table space that is to be recovered. The DSNUM option of the RECOVER utility may be used to specify:

- ▶ A partition of a partitioned table space
- ▶ A partition of a partitioned index, including a data-partitioned secondary index
- ▶ A data set within a non-partitioned table space

However, DSNUM may not be specified (at the index level) for:

- ▶ A single data set of a non-partitioned index
- ▶ A logical partition of a non-partitioned index

Alternatively, the option can recover the entire table space or index space. Example 10-7 shows the RECOVER statement for recovering four data sets in a database. Each of the four partitions will be restored in parallel. You can also run the recovery of these data sets in four separate jobs. It is better to recover all four partitions in one RECOVER statement because then all four partitions will be processed at the same time during LOGAPPLY in one pass of the logs.

*Example 10-7 Recover four data sets using DSNUM*

---

```
RECOVER PARALLEL (4)
TABLESPACE DSN8D91A.DSN8S91E DSNUM 1
TABLESPACE DSN8D91A.DSN8S91E DSNUM 2
TABLESPACE DSN8D91A.DSN8S91E DSNUM 3
TABLESPACE DSN8D91A.DSN8S91E DSNUM 4
```

---

### 10.9.1 Concurrency and compatibility

DB2 treats individual data and index partitions as distinct target objects. Utilities that operate on different partitions of the same table space or index space are compatible. However, if a non-partitioned secondary index exists on a partitioned table space, utilities that operate on different partitions of a table space can be incompatible because of contention on the non-partitioned secondary index.

When partitions of a DPSI are recovered, each partition being recovered is placed in the restrictive state DA/UTUT, where DA (Drain All) causes draining of all claim classes (that is, no concurrent SQL access is possible), and UTUT represents the utility restrictive state where the utility has exclusive control on the target object. The concurrency and compatibility characteristics for DPSIs are the same as those for PIs.

## 10.9.2 Adding partitions and recovery

Adding partitions to a partitioned table does not affect the ability to do point-in-time recovery of the object in any way. When you recover the table space to a point in time before the partition was added, the new partition is not deleted; it is only emptied.

## 10.9.3 Rotating partitions and recovery

It is important to know that recovery to a previous point in time is blocked after running the rotate statement. You cannot recover to a point in time before the ALTER TABLE ROTATE PARTITION was executed. The RECOVER fails with MSGDSNU556I and RC8. This is because all SYSCOPY and SYSLGRNX entries are deleted when the rotate statement is executed. In addition, DB2 does not support an “undo” of the rotate statement itself. After the rotate, DB2 has no knowledge of what the original limit key for the rolled-off partition was.

## 10.9.4 Rebalancing partitions with ALTER, REORG, and recovery

You can recover the partition to the current time if a recovery base (for example, a full image copy, a REORG LOG YES operation, or a LOAD REPLACE LOG YES operation) exists.

You can recover the partition to a point in time after the ALTER.

You can recover the partitions that are affected by the boundary change to a point in time prior to the ALTER; RECOVER sets REORG-pending status on the affected partitions and you must reorganize the table space or range of partitions. All affected partitions must be in the recovery list of a single RECOVER statement.

Image copies that were taken prior to resetting the REORG-pending status of any partition of a partitioned table space are not usable for recovering to a current RBA or LRSN. Determine an appropriate point in time by running REPORT RECOVERY and select an image copy for which the recovery point is a point after the rebalancing REORG was performed. If all pertinent partitions are included, the recover is successful, but the partitions will be in the REORG-pending state when the recover completes.

You can use RECOVER LOGONLY to recover data after a REORG job redistributes the data among partitions. For a point-in-time recovery, keep the offline copies synchronized with the SYSCOPY records. Do not delete, with the MODIFY RECOVERY, any SYSCOPY records with an ICTYPE column value of 'A', because these records might be needed during the recovery. Delete them only when you are sure that you no longer need to use the offline copies that were taken before the REORG that performed the rebalancing.

## 10.10 Performance improvements for RECOVER with concurrent copies

RECOVER can group the objects for dss RESTORE when the objects were copied as a group during the concurrent copy, which required that the FILTERDDN option was used at copy time.

The keyword CURRENTCOPYONLY enables the RECOVER utility to execute faster recovery when the image copies were taken using the concurrent copy feature. When you specify CURRENTCOPYONLY for a concurrent copy, RECOVER builds a DFSMSdss RESTORE command for each group of objects that is associated with a concurrent copy data set name.

RECOVER can then improve the performance of restoring concurrent copies (copies that were made by the COPY utility with the CONCURRENT option) by using only the most recent primary copy for each object in the list.

When the grouping of the objects for restore to the previous image copy data set fails, the CURRENTCOPYONLY option can drive the RESTORE behavior (to group objects).

To avoid the 255 object limit per DFSMSdss RESTORE command, the FILTERDDN option is used “under the covers” whenever required. For that purpose, a temporary file is allocated by the RECOVER utility, taking advantage of the new VOLTDEVT DSNZPARM, a unit name that is used for temporary allocations.

If the RESTORE fails, RECOVER does not automatically use the next most recent copy or the backup copy. If you specify DSNUM ALL with CURRENTCOPYONLY and one partition fails during the restore process, the entire utility job on that object fails. If you specify CURRENTCOPYONLY and the most recent primary copy of the object to be recovered is not a concurrent copy, DB2 ignores this keyword.

For objects in the recovery list whose recovery base is a system-backup, the default is CURRENTCOPYONLY. You can use REPORT RECOVERY output to determine whether the objects to be recovered have image copies, concurrent copies, or a utility log yes event. You cannot specify the ERROR RANGE option if you are recovering from a concurrent copy.

**Tip:** Do not mix objects to be restored from image copies and objects to be restored from concurrent copies in the same RECOVER utility statement.

## 10.11 LOGONLY

A DB2 object is recovered with the LOGONLY<sup>1</sup> option in the following instances:

- ▶ A backup of the object is restored outside DB2.
- ▶ A Concurrent COPY of the object is restored outside of DB2’s control.

A copy must be restored to the object prior to LOGONLY.

You cannot use RECOVER with the LOGONLY keyword for NOT LOGGED table or index spaces.

### 10.11.1 Copy taken outside DB2

Recovery with the LOGONLY keyword is only required if the DB2 object (table space or index space) is restored from a backup copy that was not created by DB2 utilities, such as COPY, LOAD, or REORG. Therefore, there is no entry in SYSCOPY. One such scenario is when the DB2 objects are backed up using a DFSMSShsm volume dump.

The external backup of the object must have been done when one or more of the following conditions is true:

- ▶ The DB2 subsystem is inactive.
- ▶ The object is closed using the **STOP DATABASE(db) SPACE(space-name)**.
- ▶ The object was QUIESCED just prior to the offline backup with WRITE YES.

<sup>1</sup> RECOVER with the LOGONLY option uses HPGRBRBA in the header page to determine where LOGAPPLY will start.

This ensures that the restore of the object is to a point of consistency, DB2 will check that the data set identifiers (DBID, PSID, and OBID) match those in the DB2 catalog. RECOVER will fail with message DSNU548I if the identifiers do not match. RECOVER applies only log records to the data set that were written after the point that is recorded in the data set page header. RECOVER will skip any image copies that were made prior to this point of consistency. The object will be recovered to the current point in time unless the TORBA or TOLOGPOINT keyword is specified in the RECOVER statement in combination with LOGONLY keyword.

Perform these steps to recover objects from backups created offline:

1. Stop the object using this command:  
`-STOP DB(db) SPACE(space)`
2. Restore the object that needs to be recovered. This will overwrite the contents of the object space with the backup data set contents.

**Important:** Be aware of the instance node of the VSAM data set during the restore (refer to 10.11.2, “DB2 object names with REORG and FASTSWITCH” on page 271). If the DB2 object is reorganized with FASTSWITCH, the node may have changed, to either I or J, as indicated by the IPREFIX column (see Table 10-2). If the object is backed up with DFSMSHsm, then the data set will be restored with the same name as the VSAM cluster at the time of the backup. The DFSMSHsm restored data set needs to be manually renamed to the correct VSAM cluster name, as indicated by the IPREFIX value.

For CLONE tables, The INSTANCE of the base or clone is reflected in the VSAM data set name as 'I0002', 'J0002', 'I0001', or 'J0001'.

3. Start the object for utility only with the command:  
`-START DB(db) SPACE(space) ACCESS(UT)`
4. To recover the object to current point in time, run the RECOVER utility with the command:  
`RECOVER TABLESPACE db.ts LOGONLY`
5. To recover to a point in time, run the RECOVER utility with the commands:  
`RECOVER TABLESPACE db.ts LOGONLY TORBA X'byte-string' or`  
`RECOVER TABLESPACE db.ts LOGONLY TOLOGPOINT X'byte-string'`
6. If recovering a table space to point in time, then the related indexes need to be rebuilt with the REBUILD utility:  
`REBUILD INDEX(ALL) TABLESPACE db.ts`
7. Allow access to the recovered object with the command:  
`-START DB(db) SPACE(space) ACCESS(RW)`
8. An image copy of the object with the COPY utility is recommended.

The TORBA point can be obtained from:

- ▶ SYSCOPY
- ▶ BSDS when the archive log is in mode quiesce
- ▶ REPORT RECOVERY report
- ▶ ssidMSTR address space DSNJ099I message
- ▶ QUIESCE job output
- ▶ Copy SHRLEVEL REFERENCE

### 10.11.2 DB2 object names with REORG and FASTSWITCH

Prior to V7, all physical data sets had the naming convention `hlq.DSNDBC.db.ts.I0001.Annn`, where the I0001 indicates the node name. DB2 V7 introduced new columns in the catalog to record the node name of the physical VSAM data set for table spaces and index spaces, as listed in Table 10-2.

Table 10-2 DB2 catalog table entries

DB2 table	Column name	Valid Values
SYSINDEXPART	IPREFIX	I or J
SYSTABLEPART	IPREFIX	I or J
SYSCOPY	STYPE	C or J for Concurrent COPY only

If the DB2 object is reorganized using the REORG SHRLEVEL CHANGE or SHRLEVEL REFERENCE with FASTSWITCH option, the catalog records the node name of the DB2 object in SYSTABLEPART or SYSINDEXPART. When the DB2 object is image copied using the COPY utility and CONCURRENT keyword, the STYPE field in SYSCOPY is recorded with the value C (I0001) or J (J0001).

### 10.11.3 Recovering a DB2 object from CONCURRENT COPY

The COPY utility with the CONCURRENT keyword invokes DFSMSdss to create image copies and update SYSCOPY with ICTYPE=F and STYPE=C or J. Refer to Chapter 9, “Copying data” on page 223 for details about the COPY utility. The concurrent image copy must have been made with the table space in a consistent state. CONCURRENT COPY of DB2 object with an 8 KB, 16 KB, or 32 KB page size must be done with SHRLEVEL REFERENCE if page size is not equal to CI size. The COPY utility initially records ICTYPE=Q when a concurrent image copy is taken with SHRLEVEL REFERENCE. When the concurrent image copy completes successfully, RECOVER changes ICTYPE=Q to ICTYPE=F.

The DB2 object can be recovered from CONCURRENT COPY in a similar manner to any recovery with ICTYPE=F. The object can be recovered as TOCOPY, TOLOGRBA, or current point in time. DB2 will apply the logs after the object is restored from the copy and rolled forward to the current point in time. If you use TOCOPY with a particular partition or data set (identified with DSNUM), the image copy must be for the same partition or data set, or for the whole table space or index space. If you use TOCOPY with DSNUM ALL, the image copy must be for DSNUM ALL. You cannot specify TOCOPY with a LIST specification.

RECOVER handles the VSAM data set node names automatically (see Table 10-2) if the object is reorganized with FASTSWITCH after the CONCURRENT COPY. During the recovery, the TOCOPY utility redefines the VSAM cluster to the correct IPREFIX node name.

**Tip:** If your shop takes both image copies and concurrent copies (that is, the COPY utility is invoked with the CONCURRENT option), recover the objects that will be restored from image copies in a separate RECOVER utility statement than the objects that will be restored from concurrent copies for performance reasons. Even if the PARALLEL option is specified, the RECOVER will use the main task to restore concurrent copies, and this could adversely affect the elapsed time of the restore of the image copies being done in parallel (by subtasks), because the main task is busy with the concurrent copies.

**Note:** The RECOVER keywords PAGE or ERRORRANGE cannot be used with DFSMS CONCURRENT COPY. If these keywords are specified, RECOVER will bypass any Concurrent COPY records when searching the SYSCOPY table for a recoverable point.

#### 10.11.4 Recovering without an image copy

When a recovery of an object is attempted where there is no record of an image copy in SYSCOPY, then one of the following actions will occur

- ▶ You will receive an error message and the recovery will fail:  
DSNU510I - NO GOOD FULL IMAGE COPY DATA SET FOR RECOVERY OF TABLESPACE.....
- ▶ You will receive an error message and the recovery will fail:  
DSNU528I - NO FULL IMAGE COPY WAS AVAILABLE AND THERE ARE NO UPDATES TO APPLY FROM THE DB2 LOG FOR TABLESPACE.....
- ▶ The object will be recovered from the HPGRBRBA if any updates are recorded in the log since the last HPGRBRBA action (refer to 10.11.6, “LOGONLY recovery improvements” on page 272).

#### 10.11.5 Recovering a single piece of a multi-linear page set

A single piece of an object (A001, A002, and so on) from a multi-piece linear page set can be recovered with the LOGONLY option. RECOVER opens the first piece of the page set to obtain the value of HPGRBRBA. If the data set is migrated by DFSMSshm, then the data set is recalled by DFSMSshm. Without LOGONLY, no data set recall is requested.

**Note:** Backing up a single piece of a multi-linear page set is not recommended. It can cause a data integrity problem if the backup is used to restore only a single data set at a later time.

**Consideration:** You cannot recover a piece of a linear table space to a point in time when the table space was in basic row format and is now in reordered row format, or vice versa. The entire table space must be recovered.

#### 10.11.6 LOGONLY recovery improvements

Prior to DB2 V7, the field HPGRBRBA, the RECOVER base RBA field on the pageset header page, representing the starting log RBA value for a log-only recovery, was updated when:

- ▶ A pseudo-close or close happened for an object.
- ▶ A stop command was issued for the object.
- ▶ A QUIESCE, LOAD utility was run against an object.

Since DB2 V7, the HPGRBRBA value is updated every *n*th checkpoint, where *n* is controlled by the value of the DSNZPARM parameter DLDFREQ (down-level detection.)

## 10.12 LOGAPPLY considerations

For the best performance of the LOGAPPLY phase, the following items should be considered:

- ▶ Try to RECOVER a list of objects in one utility statement so that you only have to take a single pass of the log. When you specify all of these objects in one statement, the utility needs to make only one pass of the log for all objects during the LOGAPPLY phase and can use parallelism when restoring the image copies in the RESTORE phase. Thus, these objects are recovered faster.
- ▶ To have your objects in R/W status after a recovery to a prior point in time, recover related sets of objects in the same RECOVER utility statement using the TOLOGPOINT option.
- ▶ Use of Fast Log Apply can reduce elapsed time in the LOGAPPLY phase. The RECOVER utility automatically uses the Fast Log Apply process during the LOGAPPLY phase if Fast Log Apply has been enabled on the DB2 subsystem.
- ▶ Keeping archive logs on disk provides the best possible performance.
- ▶ Consider the size and number of active log data sets. DB2 compares the recovery RBA to the STARTRBA and ENDRBA of the active logs. If the recovery RBA is held within the current active logs, recovery will be faster. This is due to active logs being always on disk. We recommend that you maintain at least half to a full day of log records in active logs.
- ▶ If the start log RBA is beyond the active logs, then DB2 will apply the log records from archive logs. Thus, controlling archive log data sets through DFSMSHsm is beneficial. DB2 optimizes recall of the data sets, and after being recalled, the data sets are read from disk.
- ▶ If the archive log must be read from tape, DB2 optimizes access by means of ready-to-process and look-ahead-mount requests.
- ▶ If the data sets are read from tape, set both the COUNT and the TIME values to the maximum allowable values within your system constraints. With SET ARCHIVE COUNT, you dynamically change the maximum number of input tape units to read the archive logs. With SET ARCHIVE TIME, you dynamically change the amount of time that DB2 delays deallocation.

## 10.13 Fast Log Apply

Fast Log Apply was introduced in DB2 V6 to improve the restart of DB2 subsystems after a failure and the recovery of DB2 objects. It reduced both the elapsed time and the CPU time by utilizing an intermediate buffer storage area that reduces the I/O to the table spaces.

The Fast Log Apply (FLA) design consists of one log read task per recovery job and one or more LOGAPPLY tasks employing the use of multi-tasking whenever possible. It also takes advantage of a list prefetch technique to nearly eliminate duplicate synchronous read operations for the same page. The parallel log processing for Fast Log Apply is not dependent on whether you specify RECOVER TABLESPACE name DSNUM 1 TABLESPACE name DSNUM 2 and so on, or only RECOVER TABLESPACE name, because the log apply is always done at the partition level. If you have copies at the partition level, you cannot specify RECOVER TABLESPACE dbname.tsname; you must specify RECOVER TABLESPACE dbname.tsname DSNUM 1 TABLESPACE dbname.tsname DSNUM 2 and so on. You can simplify this specification by using LISTDEF with PARTLEVEL if all parts must be recovered.

Fast Log Apply achieves these efficiencies by sorting groups of log records together that pertain to the same page set before applying those changes; it then applies those records in parallel using several LOGAPPLY tasks.

DB2 uses FLA during the following processes:

- ▶ Forward phase of DB2 restart
- ▶ RECOVER utility
- ▶ RESTORE utility
- ▶ START DATABASE for:
  - LPL recovery
  - GRECP recovery

Fast Log Apply is generally activated during the installation process by setting any non-zero value to LOGAPPLY Storage in the DSNTIPL panel, as shown in Figure 10-3. This sets a value to the DSNZPARM variable LOGAPSTG in the DSN6SYSP macro.

```

DSNTIPL          INSTALL DB2 - ACTIVE LOG DATA SET PARAMETERS
==>  _
Enter data below:

 1 NUMBER OF LOGS          ==> 3          Data sets per active log copy (2-31)
 2 OUTPUT BUFFER           ==> 4096000     Size in bytes (40K-400000K)
 3 ARCHIVE LOG FREQ         ==> 24         Hours per archive run
 4 UPDATE RATE              ==> 3600       Updates, inserts, and deletes per hour
 5 LOG APPLY STORAGE        ==> 0M         Maximum ssnmDBM1 storage in MB for
                                           fast log apply (0-100M)
 6 CHECKPOINT FREQ          ==> 50000      Log records or minutes per checkpoint
 7 FREQUENCY TYPE           ==> LOGRECS    CHECKPOINT FREQ units. LOGRECS, MINUTES
 8 UR CHECK FREQ            ==> 0          Checkpoints to enable UR check. 0-255
 9 UR LOG WRITE CHECK        ==> 0K        Log Writes to enable UR check. 0-1000K
10 LIMIT BACKOUT            ==> AUTO       Limit backout processing. AUTO,YES,NO
11 BACKOUT DURATION         ==> 5          Checkpoints processed during backout if
                                           LIMIT BACKOUT = AUTO or YES. 0-255
12 R0 SWITCH CHKPTS         ==> 5          Checkpoints to read-only switch. 1-32767
13 R0 SWITCH TIME           ==> 10        Minutes to read-only switch. 1-32767
14 LEVELID UPDATE FREQ      ==> 5          Checkpoints between updates. 0-32767

PRESS:  ENTER to continue  RETURN to exit  HELP for more information

```

Figure 10-3 DB2 installation screen DSNTIPL

**Note:** The online REORG utility uses LOGAPPLY with a different method.

### 10.13.1 Fast Log Apply buffers

DB2 obtains FLA buffers on behalf of recovery or restart tasks or during START DATABASE processing. The initial buffer size allocated depends on the particular process, as summarized in Table 10-3.

Table 10-3 Fast Log Apply buffer sizes

Process	Allocated buffer size in megabytes
RECOVER utility START DATABASE	10
DB2 RESTORE	500
DB2 Restart	10 - 100



If the storage size specified is not available when the FLA feature attempts to allocate it, DB2 tries to allocate a smaller amount. It is possible that insufficient space is available for FLA to continue; in this case, DB2 reverts to normal log recovery. A minimum of 64 KB is necessary for the FLA feature.

FLA is a non-optional feature for DB2 restart and RESTORE even if you specify zero for the Fast Log Apply storage. A minimum buffer size of 10 MB is allocated during the Forward Log Recovery phase and is freed when this phase completes. The rationale behind this decision is that restart is the only process running and there should be no problem obtaining 10 MB of storage for the FLA feature at this time. Of course, if you choose a LOGAPPLY storage value larger than 10 MB, DB2 uses that value during restart.

Each RECOVER utility job uses a size of 10 MB, if available, for the Fast Log Apply buffer. You need to consider the number of concurrent RECOVER utility jobs when determining the size of the FLA buffer. If you specify 30 MB for the total size of the LOGAPPLY buffer, then only three concurrent jobs can use 10 MB each.

DB2 acquires FLA buffer storage at the start of the Log Apply phase and releases it at the end of the Log Apply phase of the RECOVER utility. If there are four concurrent RECOVER utility jobs running, the first three jobs get 10 MB each. If the fourth job arrives at the Log Apply phase while the other three jobs are also in the Log Apply phase, then the fourth job is not able to use the FLA feature. However, recovery of the fourth job is not delayed; its recovery simply occurs without FLA.

If the fourth job arrives, at the Log Apply phase, at the same time that one of these three jobs completes the Log Apply phase, then the fourth job may be able to get the 10 MB storage, and can then also take advantage of FLA.

In data sharing, you can split the jobs across the members if there is a need to run more than 10 recoveries in parallel.

### 10.13.2 Sort the log records

The log records are sorted in DBID, OBID, PART, PAGE, and LRSN or RBA sequence. The actual sorting is accomplished by a separate task for each RECOVER and START DATABASE command as well as during restart.

Each sorted FLA buffer is divided into several smaller groups organized by the object level (DBID.PSID.PART).

One task is dispatched to apply those log records associated with each group. Each task opens the data set, if necessary, and builds and initiates the list prefetch. If the process requires the use of multiple tasks, they are run in parallel.

A maximum of 98 parallel tasks can be dispatched, one for each page set. If log records for more than 98 objects are found, then 98 tasks are scheduled. As a task completes, a new task is dispatched to apply the log records to the next object.

In addition, FLA takes advantage of the ability to perform up to 20 data set opens in parallel during restart.

### 10.13.3 When is Fast Log Apply bypassed

DB2 bypasses FLA automatically in the following situations:

- ▶ Single page recovery.
- ▶ Online page recovery.
- ▶ A minimum storage of 64 KB is not available.
- ▶ The log apply storage value is set to zero (for RECOVER utility only).

### 10.13.4 FLA performance

An increase in the number of active log buffers to 120 improves performance during the Fast Log Apply phase. Measurements show that Fast Log Apply throughput can increase up to 100% compared to DB2 V8. Fast Log Apply throughput is highest when the active log files are striped and most of the log records that are being read are inapplicable to the object or objects that are being recovered or reorganized.

When DB2 is reading archive log files, two BSAM buffers allow double buffering, but BSAM compression also allows for better I/O, especially if the archive log files are striped.

## 10.14 Avoiding specific image copy data sets

You might want to avoid a specific image copy data set for recovery. You can use the `RESTOREBEFORE` option and specify the RBA or LRSN of the image copy, concurrent copy, or system-level backup that you want to avoid, and `RECOVER` will search for an older recovery base. The `RECOVER` utility then applies log records to restore the object to its current state or the specified `TORBA` or `TOLOGPOINT` value.

## 10.15 Restarting RECOVER

You can restart `RECOVER` from the last commit point (`RESTART(CURRENT)`) or the beginning of the phase (`RESTART(PHASE)`). By default, DB2 uses `RESTART(CURRENT)`.

## 10.16 Displaying progress of RECOVER

The `DB2 -DIS LOG` command displays the archive log data sets that are allocated and being used. For a data sharing group, this command should be issued on each member.

The `-DISPLAY UTILITY` command has been enhanced so that during the `LOGAPPLY` phase of `RECOVER`, a new message `DSNU116I` will be issued. This new message indicates the range of the log that needs to be applied for the list of objects being recovered. It also shows the progress that has been made up to the last commit and the elapsed time since the start of the Log Apply phase of the recovery.

From these values, an estimate can be made as to how much longer there is to perform the recovery, as shown in Example 10-8.

*Example 10-8 DSNU116I message*

---

```
DSNU116I csect-name RECOVER LOGAPPLY PHASE DETAILS:
          STARTING TIME = timestamp
          START RBA = ss START LRSN =rr
          END RBA = ee END LRSN = nn
          LAST COMMITTED RBA = cc LAST COMMITTED LRSN = ll
          ELAPSED TIME = hh:mm:ss
```

---

## 10.17 Index recovery from COPY

Indexes of a table can be image copied using the COPY utility by altering the index space and setting COPY YES. For example:

```
ALTER INDEX creator.index-name COPY YES
```

This sets the COPY field in SYSIBM.SYSINDEXES to Y for YES. Another catalog field COPYLRSN in the same table records the RBA or the LRSN (in data sharing) of the index when it was altered to COPY YES or created with COPY YES.

One of the main advantages of the image copy index is the ability to recover the index independently from the table space or in parallel to the recovery of the table space. The latter saves an extra step to rebuild the index after the table space is recovered. In Example 10-9, we take an image copy of the table space U7G01T11.TSLINEI and all the indexes of this table space. The table space has three indexes: one partitioned and two NPIs. The COPY utility dispatched six parallel tasks to copy the objects. The maximum number of tasks is constrained by virtual storage and by available threads specified by CTHREAD and IDBACK in DSNZPARM.

*Example 10-9 COPY table space and all indexes*

---

```
//STEP1 EXEC DSNUPROC,SYSTEM=DB2G,UID=PAOL0R1,
//      UTSTATS=''
//SYSIN DD *
TEMPLATE LOCALDDN
      DSN(PAOL0R1.&DB..&TS..P&PA..T&TIME.L)
      UNIT(SYSDA) CYL DISP(MOD,CATLG,DELETE)
      VOLUMES(SBOX60)
LISTDEF DB01A
INCLUDE TABLESPACE U7G01T11.TSLINEI PARTLEVEL
INCLUDE INDEXSPACES TABLESPACE U7G01T11.TSLINEI PARTLEVEL
COPY LIST DB01A FULL YES PARALLEL 10
      SHRLEVEL CHANGE
      COPYDDN(LOCALDDN)
```

---

The recovery of the table space is done as shown in Example 10-10. We are attempting to recover the table space TSLINEI and all the indexes associated to the tables in the table space.

*Example 10-10 RECOVER of the table space and indexes in parallel*

---

```
//STEP1    EXEC DSNUPROC,SYSTEM=DB2G,UID=PAOLR1,
//          UTSTATS=' '
//SYSIN    DD *
LISTDEF DB01A
INCLUDE TABLESPACE U7G01T11.TSLINEI PARTLEVEL
INCLUDE INDEXSPACES TABLESPACE U7G01T11.TSLINEI PARTLEVEL
RECOVER LIST DB01A PARALLEL 10
```

---

The output in Example 10-11 shows that seven parallel tasks were dispatched to recover the table space, three partition indexes and two NPIs. All objects will be recovered to a point in time by applying the logs after the completion of the recovery from image copies.

*Example 10-11 RECOVER table space and indexes output*

---

```
DSNU000I  DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = PAOLR1
DSNU050I  DSNUGUTC - LISTDEF DB01A INCLUDE TABLESPACE U7G01T11.TSLINEI PARTLEVEL INCLUDE INDEXSPACES TABLESPACE
U7G01T11.TSLINEI PARTLEVEL
DSNU1035I DSNUIldr - LISTDEF STATEMENT PROCESSED SUCCESSFULLY
DSNU050I  DSNUGUTC - RECOVER LIST DB01A PARALLEL 10
DSNU397I  DSNUCBMT - NUMBER OF TASKS CONSTRAINED BY CONNECTIONS
DSNU427I  DSNUCBMT - OBJECTS WILL BE PROCESSED IN PARALLEL,
                NUMBER OF OBJECTS = 7

DSNU532I  DSNUCBMT - RECOVER TABLESPACE U7G01T11.TSLINEI DSNUM 1 START
DSNU515I  DSNUCBAL - THE IMAGE COPY DATA SET PAOLR1.U7G01T11.TSLINEI.P00001.T213329L WITH DATE=20010625 AND
TIME=173408
                IS PARTICIPATING IN RECOVERY OF TABLESPACE U7G01T11.TSLINEI DSNUM 1
DSNU532I  DSNUCBMT - RECOVER TABLESPACE U7G01T11.TSLINEI DSNUM 2 START
DSNU515I  DSNUCBAL - THE IMAGE COPY DATA SET PAOLR1.U7G01T11.TSLINEI.P00002.T213329L WITH DATE=20010625 AND
TIME=173409
                IS PARTICIPATING IN RECOVERY OF TABLESPACE U7G01T11.TSLINEI DSNUM 2
DSNU532I  DSNUCBMT - RECOVER TABLESPACE U7G01T11.TSLINEI DSNUM 3 START
DSNU515I  DSNUCBAL - THE IMAGE COPY DATA SET PAOLR1.U7G01T11.TSLINEI.P00003.T213329L WITH DATE=20010625 AND
TIME=173507
                IS PARTICIPATING IN RECOVERY OF TABLESPACE U7G01T11.TSLINEI DSNUM 3
DSNU532I  DSNUCBMT - RECOVER INDEXSPACE U7G01T11.TESTNPI2  START
DSNU515I  DSNUCBAL - THE IMAGE COPY DATA SET PAOLR1.U7G01T11.TESTNPI2.P00000.T213329L WITH DATE=20010625 AND
TIME=173404
                IS PARTICIPATING IN RECOVERY OF INDEXSPACE U7G01T11.TESTNPI2
DSNU532I  DSNUCBMT - RECOVER INDEXSPACE U7G01T11.TESTNPI  START
DSNU515I  DSNUCBAL - THE IMAGE COPY DATA SET PAOLR1.U7G01T11.TESTNPI.P00000.T213329L WITH DATE=20010625 AND
TIME=173408
                IS PARTICIPATING IN RECOVERY OF INDEXSPACE U7G01T11.TESTNPI
DSNU532I  DSNUCBMT - RECOVER INDEXSPACE U7G01T11.PXL#OKSD DSNUM 1 START
DSNU515I  DSNUCBAL - THE IMAGE COPY DATA SET PAOLR1.U7G01T11.PXL#OKSD.P00001.T213329L WITH DATE=20010625 AND
TIME=173347
                IS PARTICIPATING IN RECOVERY OF INDEXSPACE U7G01T11.PXL#OKSD DSNUM 1
DSNU532I  DSNUCBMT - RECOVER INDEXSPACE U7G01T11.PXL#OKSD DSNUM 2 START
DSNU515I  DSNUCBAL - THE IMAGE COPY DATA SET PAOLR1.U7G01T11.PXL#OKSD.P00002.T213329L WITH DATE=20010625 AND
TIME=173400
                IS PARTICIPATING IN RECOVERY OF INDEXSPACE U7G01T11.PXL#OKSD DSNUM 2
DSNU504I  DSNUCBRT - MERGE STATISTICS FOR INDEXSPACE U7G01T11.PXL#OKSD DSNUM 1 -
                NUMBER OF COPIES=1
                NUMBER OF PAGES MERGED=2318
                ELAPSED TIME=00:00:08
DSNU532I  DSNUCBMT - RECOVER INDEXSPACE U7G01T11.PXL#OKSD DSNUM 3 START
```

```

DSNU515I  DSNUCBAL - THE IMAGE COPY DATA SET PAOLOR1.U7G01T11.PXL#OKSD.P00003.T213329L WITH DATE=20010625 AND
TIME=173425
                IS PARTICIPATING IN RECOVERY OF INDEXSPACE U7G01T11.PXL#OKSD DSNUM 3
DSNU504I  DSNUCBRT - MERGE STATISTICS FOR INDEXSPACE U7G01T11.PXL#OKSD DSNUM 2 -
                NUMBER OF COPIES=1
                NUMBER OF PAGES MERGED=2308
                ELAPSED TIME=00:00:07
DSNU504I  DSNUCBRT - MERGE STATISTICS FOR TABLESPACE U7G01T11.TSLINEI DSNUM 1 -
                NUMBER OF COPIES=1
                NUMBER OF PAGES MERGED=13595
                ELAPSED TIME=00:00:24
DSNU504I  DSNUCBRT - MERGE STATISTICS FOR INDEXSPACE U7G01T11.TESTNPI2 -
                NUMBER OF COPIES=1
                NUMBER OF PAGES MERGED=9751
                ELAPSED TIME=00:00:27
DSNU504I  DSNUCBRT - MERGE STATISTICS FOR INDEXSPACE U7G01T11.TESTNPI -
                NUMBER OF COPIES=1
                NUMBER OF PAGES MERGED=10128
                ELAPSED TIME=00:00:28
DSNU504I  DSNUCBRT - MERGE STATISTICS FOR TABLESPACE U7G01T11.TSLINEI DSNUM 2 -
                NUMBER OF COPIES=1
                NUMBER OF PAGES MERGED=13831
                ELAPSED TIME=00:00:35
DSNU504I  DSNUCBRT - MERGE STATISTICS FOR INDEXSPACE U7G01T11.PXL#OKSD DSNUM 3 -
                NUMBER OF COPIES=1
                NUMBER OF PAGES MERGED=17912
                ELAPSED TIME=00:00:29
DSNU504I  DSNUCBRT - MERGE STATISTICS FOR TABLESPACE U7G01T11.TSLINEI DSNUM 3 -
                NUMBER OF COPIES=1
                NUMBER OF PAGES MERGED=107667
                ELAPSED TIME=00:01:25
DSNU500I  DSNUCBDR - RECOVERY COMPLETE, ELAPSED TIME=00:01:28
DSNU010I  DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0

```

---

An index can also be recovered individually either from an image copy or by the REBUILD utility. The decision to image copy and recover an index will depend on the size of the index space.

**Note:** Indexes with COPY NO will not have any image copies. These indexes will not participate in the recovery. Their status will be set to RBDP. These indexes must be rebuilt using the REBUILD utility. Use OPTIONS EVENT(ITEMERROR,SKIP) to bypass indexes with COPY NO when passing a list from LISTDEF.

If you want to recover a table space and all of its indexes (or a table space set and all related indexes), use a single RECOVER utility statement that specifies TOLOGPOINT.

The partitioning of secondary indexes allows you to copy and recover indexes at the entire index or individual partition level. However, if you use COPY at the partition level, you need to use RECOVER at the partition level also. If you use COPY at the partition level and then try to RECOVER the index, an error occurs. If COPY is performed at the index level, you can use RECOVER at either the index level or the partition level.

**Consideration:**

- ▶ You cannot recover an index or index space to a PIT prior to an ALTER INDEX REGENERATE. The index should be rebuilt by using the REBUILD INDEX utility.
- ▶ You cannot recover to a point in time after ALTER TABLE ADD COLUMN and before ALTER COLUMN DROP DEFAULT.
- ▶ You cannot recover a point-in-time RECOVER INDEX if the recovery point precedes the first ALTER INDEX. In Version 8 new-function mode, this creates a new index version.
- ▶ You cannot recover an index space to a point in time that is prior to rotating partitions. After you rotate a partition, you cannot recover the contents of that partition to a point in time that is before the rotation.

## 10.18 PIT to a different DB2 function mode

You should be aware of some special considerations when you are recovering catalog, directory, and all user objects to a point in time in which the DB2 subsystem was in a different mode, for example, if your DB2 subsystem is currently in new-function mode, and you need to recover to a point in time in which the subsystem was in conversion mode.

Before you recover the DB2 catalog, directory, and all user objects to a prior point in time, you should shut down the DB2 system cleanly and then restart the system in ACCESS(MAINT) mode. Then you can recover the catalog and directory objects to a PIT following the order of recovery documented in the *DB2 Version 9.1 for z/OS Utility Guide and Reference*, SC18-9855, which can also direct you about rebuilding or recovering catalog and directory indexes. You can use the RECOVER utility to recover catalog and directory indexes if the indexes were defined with the COPY YES attribute and if you have the full index image copies.

You can then use the sample queries and documentation, provided in the DSNTESQ member of the SDSNSAMP sample library, to check the consistency of the catalog.

## 10.19 General performance recommendations for RECOVER

The following options are available to obtain the fastest possible recovery:

- ▶ Allow the RECOVER utility to restore table spaces in parallel.
- ▶ Place the image copy on disk for faster restore time.
- ▶ Take frequent image copies.
- ▶ Consider using incremental image copies.
- ▶ Consider using the MERGECOPY utility when producing incremental image copies.
- ▶ Consider using COPY and RECOVER for large indexes.
- ▶ Consider running recovery jobs in parallel, but no more than ten per DB2 subsystem or member.
- ▶ Exploit Fast Log Apply.
- ▶ Exploit Parallel Index Rebuild.
- ▶ Archive logs to disk.

- ▶ Specify log data set block size as 24576 bytes.
- ▶ Exploit the large block interface support in DB2 9 for image copies that are on tape.
- ▶ Maximize the data available from active logs.
- ▶ Increase the accuracy of the DB2 log range registration.
- ▶ Consider using I/O striping for the active logs.
- ▶ Consider using the DB2 Log Accelerator tool to provide striping and compression support for the archive logs.
- ▶ Evaluate data compression for reducing log data volumes.
- ▶ Tune your buffer pools.

## 10.20 MODIFY RECOVERY

The MODIFY utility with the RECOVERY option deletes records from the SYSIBM.SYSCOPY catalog table, related log records from the SYSIBM.SYSLGRNX directory table, and entries from the database descriptor (DBD). This option also recycles DB2 version numbers for reuse.

You can remove records that were written before a specific date or of a specific age, and you can delete records for an entire table space, partition, or data set. DB2 9 enhancements simplify the usage of MODIFY RECOVERY and make it safer to use by not deleting more than is intended.

With DB2 9, the MODIFY RECOVERY utility no longer acquires the “X” DBD lock when removing the object descriptors (OBDs) for tables that had been previously dropped. In DB2 9, the “U” DBD lock is acquired instead of the X DBD lock to allow availability and flexibility.

Another improvement to modify recovery in DB2 9 is more frequent commits during the deletion of SYSLGRNX records. This helps to improve concurrency on SYSLGRNX and to avoid timeout conditions on SYSLGRNX pages.

The MODIFY RECOVERY utility has been enhanced to allow SYSIBM.SYSLGRNX records to be deleted for a table space or index (with the COPY YES attribute), even if no SYSIBM.SYSCOPY records were deleted. This enables users of BACKUP SYSTEM and RESTORE SYSTEM utilities to clean up their SYSIBM.SYSLGRNX records, even though they do not take traditional image copies of their objects with the COPY utility. A new SYSIBM.SYSCOPY record with ICTYPE='M' and STYPE='R' is inserted by a MODIFY RECOVERY job so that the RECOVER utility can ensure that it has all of the recovery information that it needs to use system-level backups as the recovery base during object-level recoveries. MODIFY RECOVERY always inserts a SYSIBM.SYSCOPY row with ICTYPE="M" and STYPE="R" to record the RBA or LRSN (in case of data sharing) of the most recent SYSCOPY or SYSLGRNX record deleted.

A new keyword "RETAIN(n)" is added to the MODIFY RECOVERY utility control statement, which is used to determine the cleanup date. Only records with ICTYPE=F (full copy), ICBACKUP=BLANK (LOCALSITE primary copy), and DSNUM (stated for the specified table space) are checked.

Depending on the chosen option (LAST, LOGLIMIT, or GDGLIMIT), the specific records are retained in SYSIBM.SYSCOPY. The following considerations apply:

- ▶ Be careful in specifying GDGLIMIT if multiple GDG bases are used for a single object, such as in the case of full image copies using one GDG and incremental image copies using a separate GDG.
- ▶ The date and time of creating the image copy are not visible in the data set, but only in SYSIBM.SYSCOPY.
- ▶ There is no selection of specific objects when they are created on the same day.
- ▶ Dynamic alteration of database and table space names requires higher administration to define a new GDG base.
- ▶ GDG entries in SYSCOPY are not usable or empty because of copy failures, such as due to a B37.
- ▶ With respect to SAP databases, never use either GDGLIMIT as an option or generation data set (GDS) or GDGs for creating image copies.

Here are some considerations when using GDGs and SAP:

- ▶ If you use the CLONE keyword on your MODIFY RECOVERY utility statement, DB2 deletes all corresponding SYSCOPY and SYSLGRNX entries for your clone objects. This is a different behavior than when DB2 drops a clone. When you drop a clone, DB2 deletes the SYSCOPY entries, but keeps the SYSLGRNX entries.
- ▶ MODIFY RECOVERY also resets a flag kept in the OBD about ALTER ADDED columns. This flag is needed for point-in-time recovery to work correctly. As long as there are image copies that can be recovered, you need to make sure to keep the record format information for the row versions that were used when those image copies were taken. Even if the data sets might have gone already, DB2 still keeps track of those image copies in SYSCOPY, so make sure that MODIFY RECOVERY is run on a regular basis to get rid of all image copy entries after a new column was added to a table and before the next REORG that follows the ALTER that materialized the new column.
- ▶ Another reason to use the MODIFY RECOVERY utility on a weekly or daily basis is the large number of objects, for example, if there are a large number of table spaces and indexes in SAP databases. If only three backups remain in SYSIBM.SYSCOPY, the average count of image copies is about 100,000 data sets for, for example, only one SAP system. SYSIBM.SYSLGRNX and SYSIBM.SYSCOPY can become large and take up a considerable amount of space.

Some of the objects only change a few times a year. One of the goals of DB2 administrators is to avoid unnecessary image copies and reduce CPU and space impact. To face this demand, it is a good practice to obtain new image copies only when changes occur. To further improve performance and minimize lock contention, perform a REORG of both DSNDB06.SYSCOPY and SYSLGRNX table spaces on a regular basis as well.

**Tip:** Run MODIFY RECOVERY on a regular basis to keep SYSIBM.SYSCOPY and SYSIBM.SYSLGRNX in a manageable state. Use REORG on SYSIBM.SYSCOPY and SYSIBM.SYSLGRNX regularly.

If you are using BACKUP SYSTEM and not regular image copies, REORG still inserts rows in SYSIBM.SYSCOPY. In this case, we recommend that you continue to use MODIFY RECOVERY with DELETE AGE or DELETE DATE.



Figure 10-4 shows how to get eliminate the ALTER ADDED COLUMN flag and the old table descriptor. It is necessary to run MODIFY RECOVERY to delete all image copies that could contain old format records. MODIFY RECOVERY needs to delete ALL image copies before the next REORG that follows the ALTER command (or the last ALTER command). If MODIFY RECOVERY ... DELETE DATE(20080411) is invoked, then it will eliminate all image copies before the ALTER, and even one after it, but this is not sufficient, as there is one more image copy that might contain old records. This means that the ALTER ADDED COLUMN flag will not be reset with that invocation. Only when MODIFY RECOVERY is run with DELETE DATE(20080412) will that flag be reset, as all “old format” image copies will then be deleted.

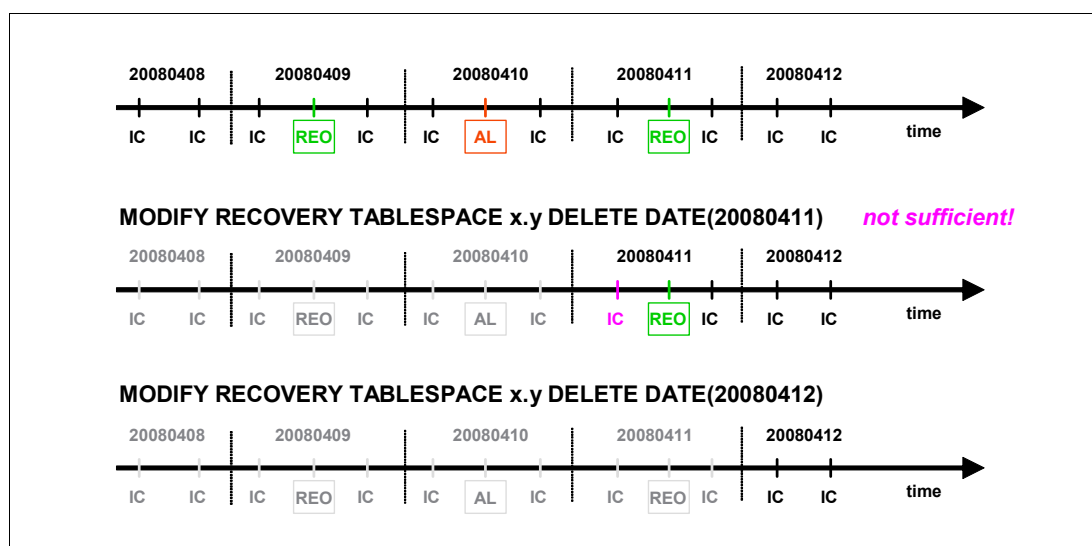


Figure 10-4 Best practices for MODIFY RECOVERY DATE

**Note:** Deletion works on dates and not on timestamps. As a result, more entries than requested might be kept. For example, if the five most recent copies were taken on the same day, and `RETAIN LAST(2)` is specified, then the records for all five copies that were taken on that day are retained in SYSCOPY.

Figure 10-5 shows a simplified MODIFY RECOVERY with the new options.

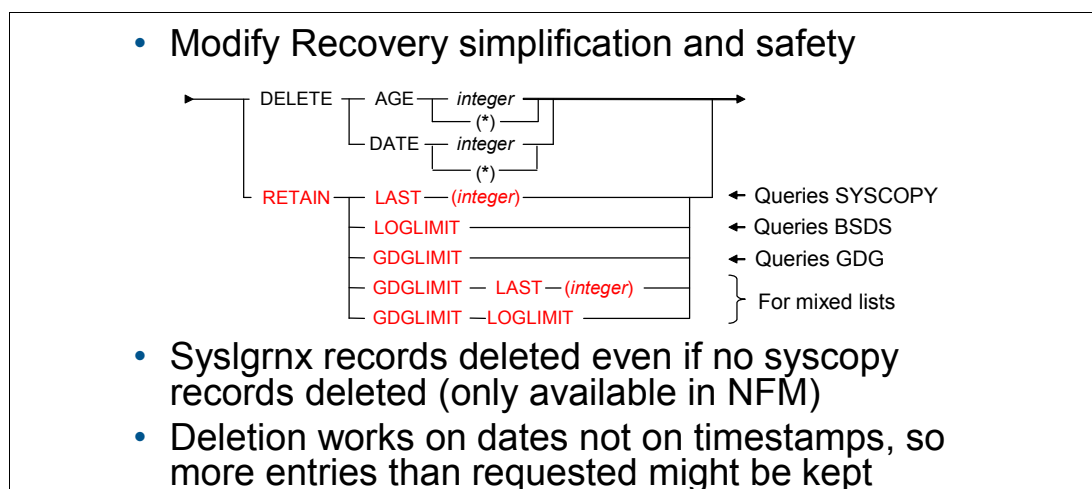


Figure 10-5 New MODIFY RECOVERY options

## 10.20.1 RETAIN LAST(n)

RETAIN LAST(n) specifies the number of recent records to retain in SYSIBM.SYSCOPY. RETAIN works with a date and not with a complete timestamp. This can result in more copies kept, as specified by RETAIN. For example, if the most recent five copies were taken on the same day and "RETAIN LAST(2)" is specified, all five copies of this day remain in SYSIBM.SYSCOPY.

Figure 10-6 illustrates how RETAIN LAST (which was introduced in DB2 9 for z/OS) works. it will always work on day boundaries. So if RETAIN LAST 3 is specified, it will delete all records before the day when the last three full image copies were taken. As the third last image copy was taken on 2008-04-11, it will delete everything before that day as though it were invoked with DELETE DATE(20080411). This will actually retain four image copies, so you should not be surprised if that happens. If invoked with RETAIN LAST 2 in this particular example, this will result in exactly that number of full image copies being retained.

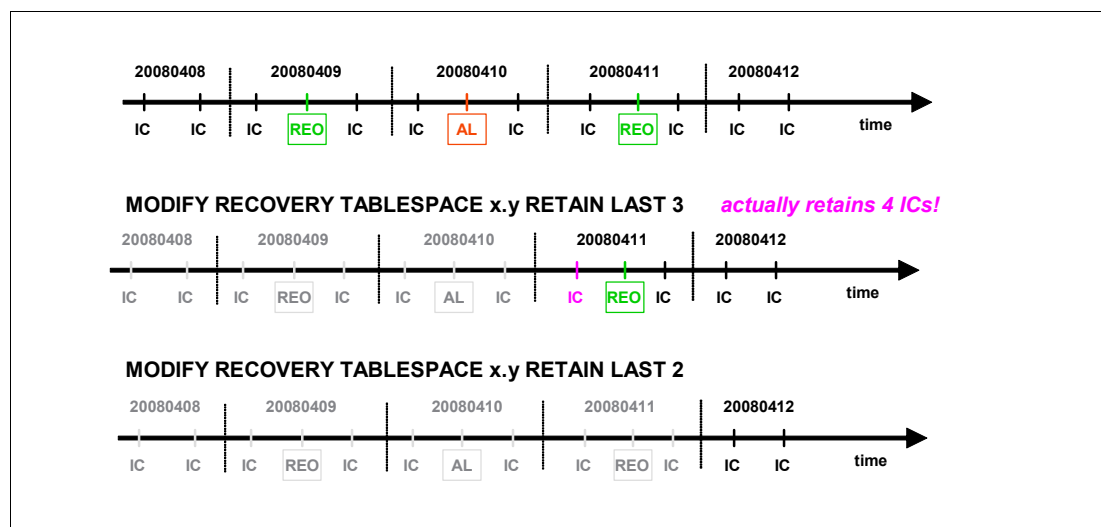


Figure 10-6 Best practices for MODIFY RECOVERY RETAIN

## 10.20.2 RETAIN LOGLIMIT

RETAIN with the LOGLIMIT option queries the BSDS to determine the oldest archive log timestamp and deletes records in SYSIBM.SYSCOPY and SYSIBM.SYSLGRNX older than the timestamp. For data sharing, DB2 queries the BSDS of all data sharing members to determine the overall and oldest log timestamp.

## 10.21 REPORT RECOVERY

REPORT RECOVERY is an online utility that can be used to find information necessary for recovering a table space, index, or table space and all its indexes. The utility also provides the LOB table spaces associated with a base table space.

REPORT does not set a utility restrictive state on the target table space or partition. REPORT can run concurrently on the same target object with any utility or SQL operation.

The output from REPORT RECOVERY consists of:

- ▶ Recovery history from the SYSIBM.SYSCOPY catalog table, including QUIESCE, COPY, LOAD, REORG, RECOVER TOCOPY, and RECOVER TORBA history
- ▶ Log ranges from the SYSIBM.SYSLGRNX directory table
- ▶ Archive log data sets from BSDS
- ▶ Any indexes on the table space that are in informational COPY-pending status

In a data sharing environment, the output provides:

- ▶ The RBA of when DB2 was migrated to Version 9.1.
- ▶ The high and low RBA values of the migrated member.
- ▶ A list of any SYSLGRNX records from before data sharing was enabled that cannot be used to recover to any point in time after data sharing was enabled.
- ▶ For SYSCOPY, the member from which the image copy was deleted.

The TABLESPACESET option can be used to find the names of all table spaces and tables in a referential structure, including LOB and XML table spaces.

With the SHOWDSNS option, you specify that the VSAM data set names for each table space or index space are to be included in the TABLESPACESET report. Data set names for base objects are shown in the section titled TABLESPACE SET REPORT. Data set names for CLONE objects are shown in the section titled CLONE TABLESPACE SET REPORT. The report is only prepared if the base objects have been cloned. With the DSNUM option, you can report information about a partition or data set.

If REPORT TABLESPACESET or REPORT RECOVERY is specified and the base objects have been cloned, information for both base and clone objects will be displayed in the output.

If you specify CLONE on the REPORT utility, you receive information for only the specified objects that are table spaces and contain clone tables, indexes on clone tables, or index spaces that contain indexes on clone tables.

If you specify CLONE together with TABLESPACESET, REPORT also processes related LOBS. Because referential constraints are not allowed for tables with clones, TABLESPACESET does not list dependent table objects for your clone tables.

You need to specify the CURRENT option to avoid unnecessary access of archive logs. If archive logs are on tape, then there will be unnecessary mounting of archive tapes. If the CURRENT option is specified and the last recoverable point does not exist on the active log, DB2 will access archive logs until the point is found.

The REPORT RECOVERY utility shows new SYSIBM.SYSCOPY column TTYPE values. Table 10-4 shows the new values.

Table 10-4 NEW TTYPE values

ICTYPE	STYPE	Inserted by
E	M,C,L, blank	RECOVER to CURRENT
A	O	ALTER to NOT LOGGED
A	L	ALTER to LOGGED
M	R	MODIFY RECOVERY
A	E	EXCHANGE DATA

### **Sample REPORT RECOVERY job**

Example 10-12 is a sample JCL that reports on table space DSN8S71D. The CURRENT option is specified to report only the SYSCOPY entries after the last recoverable point of the table space.

#### *Example 10-12 REPORT RECOVERY sample job*

---

```
//STEP1    EXEC DSNUPROC,SYSTEM=DB2G,UID=REPORT
//SYSIN    DD  *
LISTDEF DB01A
          INCLUDE TABLESPACE DSN8D71A.DSN8S71D
REPORT RECOVERY TABLESPACE
          LIST DB01A CURRENT ARCHLOG 1
```

---

Example 10-13 is the output of the sample REPORT RECOVERY job. The output shows that a full image copy exists for the table space DSN8S71D at START LRSN 00010DEA81F9. The associated entry in SYSLGRNX shows that the table space DSN8S71D was open at UCDATE and UCTIME and at START LRSN 00010DF08B11. The 000000000000 value for STOP LRSN indicates that the table space is still open.

#### *Example 10-13 REPORT RECOVERY sample output*

---

```
DSNU000I  DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = CHECKIX
DSNU050I  DSNUGUTC - LISTDEF DB01A INCLUDE TABLESPACE DSN8D71A.DSN8S71D
DSNU1035I DSNUIHDR - LISTDEF STATEMENT PROCESSED SUCCESSFULLY
DSNU050I  DSNUGUTC - REPORT RECOVERY TABLESPACE LIST DB01A CURRENT ARCHLOG 1
DSNU1033I DSNUGULM - PROCESSING LIST ITEM: TABLESPACE DSN8D71A.DSN8S71D
DSNU581I -DB2G DSNUPREC - REPORT RECOVERY TABLESPACE DSN8D71A.DSN8S71D
DSNU585I -DB2G DSNUPREC - REPORT RECOVERY TABLESPACE DSN8D71A.DSN8S71D CURRENT
DSNU593I -DB2G DSNUPREC - REPORT RECOVERY ENVIRONMENT RECORD:
'          MINIMUM RBA: 000000000000
'          MAXIMUM RBA: FFFFFFFFFF
'          MIGRATING RBA: 000000000000
DSNU582I -DB2G DSNUPPCP - REPORT RECOVERY TABLESPACE DSN8D71A.DSN8S71D SYSCOPY ROWS
TIMESTAMP = 2001-06-28-20.04.50.300816, IC TYPE = F , SHR LVL = C, DSNUM   = 0000, START LRSN
=00010DEA81F9
DEV TYPE = 3390      , IC BACK =      , STYPE =      , FILE SEQ = 0000, PIT LRSN =
000000000000
LOW DSNUM = 0000, HIGH DSNUM = 0000
JOBNAME = PAOLOR1C, AUTHID = PAOLOR1 , COPYPAGESF = 3.0E+00
NPAGESF = 1.2E+01 , CPAGESF = 2.0E+00
DSNAME = PAOLOR1.DSN8D71A.DSN8S71D.P00000.T000446L , MEMBER NAME =

DSNU586I -DB2G DSNUPSUM - REPORT RECOVERY TABLESPACE DSN8D71A.DSN8S71D SUMMARY
DSNU588I -DB2G DSNUPSUM - NO DATA TO BE REPORTED
DSNU583I -DB2G DSNUPPLR - SYSLGRNX ROWS FROM REPORT RECOVERY FOR TABLESPACE DSN8D71A.DSN8S71D
UCDATE   UCTIME   START RBA   STOP RBA   START LRSN   STOP LRSN   PARTITION   MEMBER ID
062801   20091248 00010DF08B11 000000000000 00010DF08B11 000000000000 0000        0000

DSNU584I -DB2G DSNUPPBS - REPORT RECOVERY TABLESPACE DSN8D71A.DSN8S71D ARCHLOG1 BSDS VOLUMES
DSNU588I -DB2G DSNUPPBS - NO DATA TO BE REPORTED

DSNU586I -DB2G DSNUPSUM - REPORT RECOVERY TABLESPACE DSN8D71A.DSN8S71D SUMMARY
DSNU588I -DB2G DSNUPSUM - NO DATA TO BE REPORTED
DSNU589I -DB2G DSNUPREC - REPORT RECOVERY TABLESPACE DSN8D71A.DSN8S71D COMPLETE

DSNU580I  DSNUPORT - REPORT UTILITY COMPLETE - ELAPSED TIME=00:00:00
DSNU010I  DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

---

The SYSCOPY and SYSLGRNX information can be used to determine the recovery details of the table space. It can also be used to decide if an image copy is required. If there is an entry START LRSN in SYSLGRNX and its value is higher than the START LRSN of the last image copy entry in SYSCOPY, then the table space is open for update. This table space is a candidate for image copy using the COPY utility. RTS can tell whether an object has been updated since the last copy.

For non-data sharing, start and end LRSN columns in SYSLGRNX now contain a store clock timestamp.

One advantage of using REPORT RECOVERY to determine image copy status is that the base table space is not accessed. All the required information is obtained from SYSCOPY and SYSLGRNX. Therefore, if the base table space is migrated by DFSMSHsm, it will not be recalled.

### ***Catalog and directory table spaces***

The SYSCOPY information is being kept in the DB2 logs, so REPORT may cause archive log data sets to be recalled or mounted when running REPORT RECOVERY for:

- ▶ SYSIBM.SYSCOPY
- ▶ SYSIBM.SYSLGRNX
- ▶ SYSIBM.DBD01

**Tip:** Always run the REPORT RECOVERY utility on each object and determine the recovery base before running the RECOVER utility.

## **10.22 DB2 Recovery Expert**

With DB2 Recovery Expert, you can select the fastest, least costly method that the most appropriate for your DB2 recovery. It enhances your ability to make the best decision when time is of the essence.

DB2 Recovery Expert is able to back up and recover data using DASD fast copy (FlashCopy) and extends the capabilities of the DB2 V8 and DB2 9 backup and restore system utilities, such as the following:

- ▶ Helps with disaster recovery by building assets that can be moved to a disaster site for rebuilding DB2, including full support of the Restore System feature in DB2 for z/OS Version 8.
- ▶ Menus for point-in-time recovery are easy to use and help you understand when to choose options such as rolling data forward or backward.
- ▶ There is a versioning repository, which includes related dependent objects, even if they no longer exist in the DB2 catalog.
- ▶ A log analysis function helps you determine “quiet time” to ensure that recovered objects do not have activity happening against them.
- ▶ Data object types stored in the DB2 catalog are supported, including database, table space, views, and table recovery.
- ▶ Recovery of object types implemented in SQL (for example, procedures and user-defined functions) are supported.

For more information, refer to “DB2 Recovery Expert for z/OS” on page 456.

## 10.23 DB2 Change Accumulation Tool

The DB2 Change Accumulation Tool creates full image copies of objects without the need to start or stop the database. By using this low-impact tool, you can minimize database downtime and reduce recovery time. For more information, refer to “DB2 Change Accumulation Tool” on page 459.

## 10.24 QUIESCE

QUIESCE is an online utility that establishes a consistent recovery point (the current log RBA or LRSN) for a table space, partition, table space set, and index with COPY YES set. QUIESCE WRITE(NO) is less disruptive because the COPY NO indexes are not quiesced. DB2 9 introduced the ability to recover to any prior point in time with the RECOVER utility. Because this ability lets you back out URs when recovering to any point in time, we suggest that you can run the QUIESCE utility less frequently to further increase your service availability.

A QUIESCE can be executed on a table space, such as DSNDB06.SYSEBCDC, which contains a table SYSIBM.SYSDUMMY, which only has one row that no one updates. QUIESCE can be run on this catalog table space to establish an RBA or LRSN value for that point in time. That QUIESCE RBA or LRSN can then be used if a *real* table space needs to be recovered to a point in time via RECOVER TOLOGPOINT. The QUIESCE on the dummy table is used to quickly establish an RBA or LRSN value for that point in time.

The only reasons to run QUIESCE now are:

- ▶ To mark, in SYSCOPY, a point in time that has significance, such as the beginning or end of major batch processing
- ▶ To create a common sync-point across multiple systems, for example, across IMS and DB2

If you specify CLONE on the QUIESCE utility, you want to create a quiesce point for only the specified table spaces that contain clone tables.

QUESCE supports a list of table spaces, as shown in Example 10-14.

*Example 10-14 QUIESCE with a list of table spaces*

---

```
//STEP1    EXEC DSNUPROC,SYSTEM=DB2G,UID=QUIESCE
//SYSIN    DD  *
QUIESCE
    TABLESPACE DSNRLST.DSNRLS01
    TABLESPACE DSNDB04.MYTABLE
    TABLESPACE DSNDB04.EXAMP2
    TABLESPACE DSN8D71P.DSN8S71Q
    TABLESPACE DSNDB04.SYSTABLE
WRITE YES
```

---

As an alternative, you can build a list using the LISTDEF command, as shown in Example 10-15.

*Example 10-15 QUIESCE using the LISTDEF command*

---

```
//STEP1      EXEC DSNUPROC,SYSTEM=DB2G,UID=QUIESCE
//SYSIN      DD *
LISTDEF DB01A
            INCLUDE TABLESPACE DSN*.*
            EXCLUDE TABLESPACE DSN8D71A.DSN8S71D
            EXCLUDE TABLESPACE DSN8D71A.DSN8S71E
            EXCLUDE TABLESPACE DSNDB04.EXAMP1
QUIESCE LIST DB01A WRITE YES
```

---

The TABLESPACESET option for QUIESCE option specifies that all of the referentially related table spaces in the table space set are to be quiesced. A table space set is either:

- ▶ A group of table spaces that have a referential relationship
- ▶ A base table space with all of its LOB or XML table spaces

The associated indexes are quiesced only if WRITE YES is specified.

**Tip:** Use QUIESCE Utility on a LISTDEF list of related objects or use TABLESPACESET.

SYSIBM.SYSCOPY is updated with ICTYPE=Q for each table space quiesced. Other enhancements and restrictions for the QUIESCE utility are:

- ▶ Remove the restriction of 1,165 maximum table spaces in the list.
- ▶ Duplicate table spaces or table space sets are quiesced only once.
- ▶ For a table space being quiesced with WRITE YES, a SYSCOPY record with ICTYPE=Q is also inserted for each of its indexes defined with COPY YES.
- ▶ PART and TABLESPACE are mutually exclusive keywords.
- ▶ An index or index space cannot be specified on a QUIESCE statement.

Example 10-16 shows the quiesce of indexes of a partitioned table space TSLINEI when only the table space is specified in the QUIESCE list.

*Example 10-16 QUIESCE of table space and all indexes*

---

```
DSNU000I  DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = CPITEM
DSNU050I  DSNUGUTC - LISTDEF DB02A INCLUDE TABLESPACE U7G01T11.TSLINEI PARTLEVEL
DSNU1035I DSNUIHDR - LISTDEF STATEMENT PROCESSED SUCCESSFULLY
DSNU050I  DSNUGUTC - QUIESCE LIST DB02A
DSNU481I -DB2G DSNQUAIA - QUIESCE SUCCESSFUL FOR TABLESPACE U7G01T11.TSLINEI PARTITION 1
DSNU481I -DB2G DSNQUAIA - QUIESCE SUCCESSFUL FOR INDEXSPACE U7G01T11.PXL#OKSD PARTITION 1
DSNU477I -DB2G DSNQUAIA - QUIESCE SUCCESSFUL FOR INDEXSPACE U7G01T11.TESTNPI
DSNU477I -DB2G DSNQUAIA - QUIESCE SUCCESSFUL FOR INDEXSPACE U7G01T11.TESTNPI2
DSNU481I -DB2G DSNQUAIA - QUIESCE SUCCESSFUL FOR TABLESPACE U7G01T11.TSLINEI PARTITION 2
DSNU481I -DB2G DSNQUAIA - QUIESCE SUCCESSFUL FOR INDEXSPACE U7G01T11.PXL#OKSD PARTITION 2
DSNU477I -DB2G DSNQUAIA - QUIESCE SUCCESSFUL FOR INDEXSPACE U7G01T11.TESTNPI
DSNU477I -DB2G DSNQUAIA - QUIESCE SUCCESSFUL FOR INDEXSPACE U7G01T11.TESTNPI2
DSNU481I -DB2G DSNQUAIA - QUIESCE SUCCESSFUL FOR TABLESPACE U7G01T11.TSLINEI PARTITION 3
DSNU481I -DB2G DSNQUAIA - QUIESCE SUCCESSFUL FOR INDEXSPACE U7G01T11.PXL#OKSD PARTITION 3
DSNU477I -DB2G DSNQUAIA - QUIESCE SUCCESSFUL FOR INDEXSPACE U7G01T11.TESTNPI
DSNU477I -DB2G DSNQUAIA - QUIESCE SUCCESSFUL FOR INDEXSPACE U7G01T11.TESTNPI2
DSNU474I -DB2G DSNQUAIA - QUIESCE AT RBA 00010DBEDA76 AND AT LRSN 00010DBEDA76
DSNU475I  DSNQUAIB - QUIESCE UTILITY COMPLETE, ELAPSED TIME= 00:00:00
DSNU010I  DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

---

### 10.24.1 DPSI drain class and states considerations

A partitioning index (PI), data-partitioned secondary index (DPSI), and partition and non-partitioned secondary indexes (NPSI) have the same drain classes and restrictive states. When running QUIESCE WRITE YES, the related partition(s) are in DW/UTRO states. There are no drains or restrictive states during QUIESCE WRITE NO against the DPSI parts. For non-partitioned secondary indexes (NPSIs), the entire index is DW/UTRO during QUIESCE WRITE YES operations against the table space or any partition of the table space.

DW (Drain Writers) means draining the write class, permitting concurrent access for SQL readers, but not allowing any updaters to access the object. UTRO represents the utility restrictive state, allowing read only access on the target object.

### 10.24.2 Informational RI and QUIESCE

An application that does the RI checking inside that application can take advantage of defining informational RI in DB2. Defining informational RI allows you to easily quiesce related objects. Using informational RI and QUIESCE TABLESPACESET gives you an easy way to quiesce a set of application related objects, without having to track them individually and making sure you include all of them in the QUIESCE utility.

### 10.24.3 QUIESCE NOT LOGGED considerations

QUIESCE on a table space with the NOT LOGGED attribute does not create a recoverable point, because there are no log records that a subsequent RECOVER utility can apply to recover the data to the quiesced point.

If QUIESCE with the WRITE(NO) option is requested on a NOT LOGGED table space, then informational message DSNU485I is issued and the object is skipped. If QUIESCE with the WRITE(YES) option is requested on a NOT LOGGED table space, the table space and its indexes are drained of writers, and the corresponding pages are written from the buffer pool to DASD.





# Gathering statistics

In this chapter we describe the use of RUNSTATS for gathering statistics on DB2 objects, and we also detail the ability to evaluate trend analysis by using the History function.

In this chapter, we discuss these topics:

- ▶ Why collect statistics
- ▶ When to use RUNSTATS
- ▶ History tables
- ▶ SAMPLE option
- ▶ KEYCARD
- ▶ FREQVAL
- ▶ COLGROUP
- ▶ LOB table space
- ▶ Performance
- ▶ Inline execution with LOAD/REORG/REBUILD INDEX
- ▶ DSTATS
- ▶ HISTOGRAM
- ▶ MODIFY statistics

## 11.1 Why collect statistics

Statistics about the data contained within DB2 objects are required for these reasons:

- ▶ To provide access path data that the optimizer can use to select the correct access path to the data for queries at either BIND or PREPARE time. Some can also be used to help determine when to REORG.
- ▶ To provide space information about the space occupied by the data and the state of the underlying data sets. These figures can be used to:
  - Determine the optimum sizing for the underlying data sets.
  - Determine how well disk space is being used.
  - Determine the correct settings for PCTFREE and FREEPAGE.
  - Determine when to reorganize the data.
  - Monitor growth for capacity planning.
  - Monitor the effectiveness of the compression dictionary.
- ▶ To improve the accuracy of the optimizer to improve the query performance for skewed data by collecting the non-uniform distribution statistics on non-indexed columns and distribution statistics on any column in your tables, so you can provide the optimizer with better filter factor information in the DB2 catalog.

Because collecting detailed information about distributions and grouping them by columns can be expensive, your collection should be limited to index columns and data columns utilized by predicates in the query. Several tools provide an advisory function for verifying the validity of statistics and suggesting the statements for collecting the “right” statistics, including the Optim Query Tuner and Optim Query Workload Tuner tools discussed in “Optim Query Tuner and Optim Query Workload Tuner” on page 460.

The statistics can also be used by DBAs to reevaluate physical database design.

## 11.2 RUNSTATS

RUNSTATS collects statistics about user table spaces, partitions, and indexes that DB2 uses for access path determination and to determine if a REORG is necessary. You can use the same statistics to determine if a REORG is needed for the user and catalog table spaces. The only difference is the information in the columns NEAROFFPOSF and FAROFFPOSF in table SYSINDEXPART. The values in these columns can be double the recommended values for user table spaces before a reorganization is needed if the table spaces are DSND06.SYSDBASE, DSND06.SYSVIEWS, DSND06.SYSPLAN, DSND06.SYSGROUP, or DSND06.SYSDBAUT.

Information from the RUNSTATS utility can tell you how well disk space is being used. You can find, for example, in the SYSTABLEPART catalog table the number of varying-length rows that were relocated to other pages because of an update. RUNSTATS maintains the related catalog columns, and then you can check the status of the table space using the statement shown in Example 11-1.

*Example 11-1 Check the status of the table space*

---

```
SELECT CARD, NEARINDREF, FARINDREF
FROM SYSIBM.SYSTABLEPART
WHERE DBNAME = 'XXX'
AND TSNAME = 'YYY';
```

---

Figure 11-1 shows what statistics are gathered by RUNSTATS.

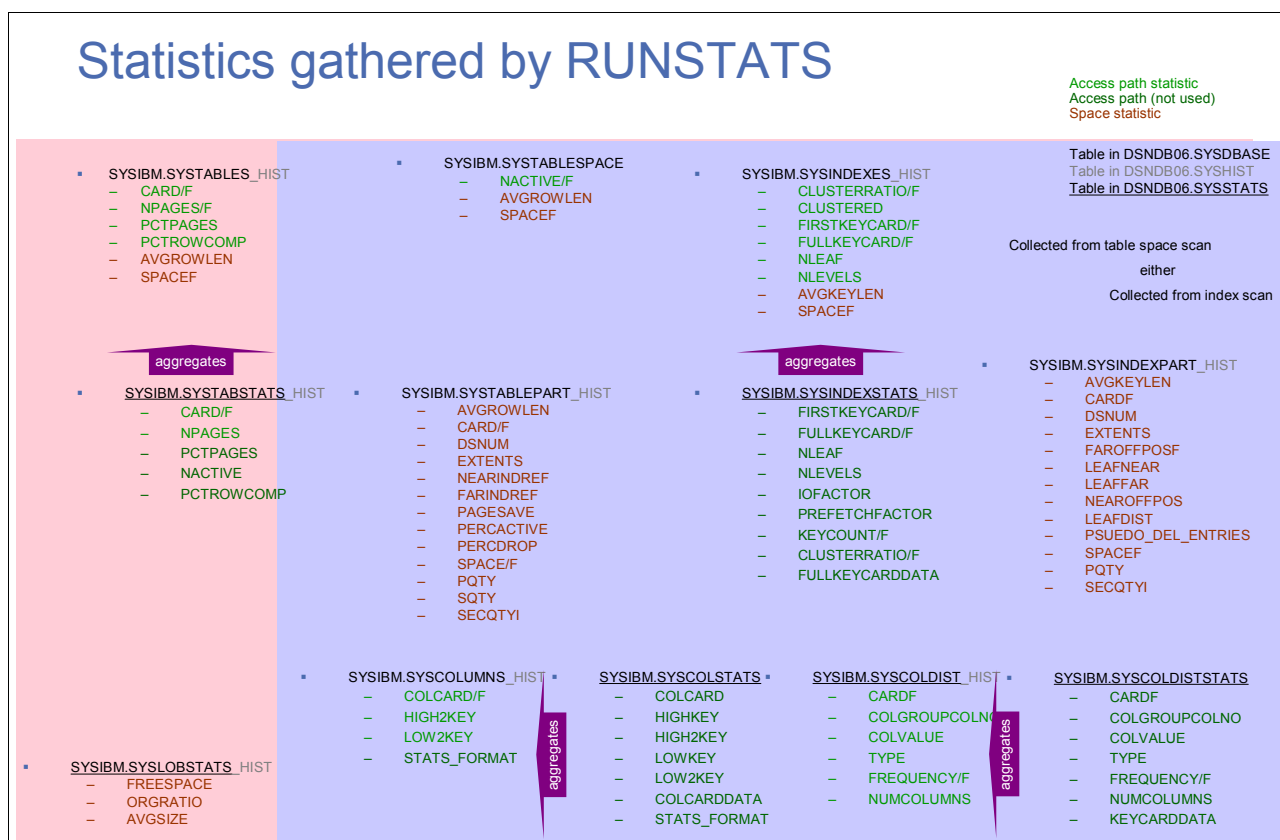


Figure 11-1 Statistics gathered by RUNSTATS

For RUNSTATS, a new DSNZPARM STATCLUS value has been added. This value specifies the default RUNSTATS clustering statistics type. The default is ENHANCED. This parameter should improve access path selection for duplicate and reverse-clustered indexes. For indexes that contain either many duplicate key values or key values that are highly clustered in reverse order, cost estimation that is based purely on CLUSTERRATIOF can lead to repetitive index scans. A new cost estimation formula based on the DATAREPEATFACTORF statistic to choose indexes avoids this performance problem. To take advantage of the new formula, set the STATCLUS parameter to ENHANCED. Otherwise, set the value to STANDARD. Additional DSNZPARM parameters OPTIOWGT =ENABLE (the default since PK75643/UK42565) enables DB2 to use a new formula that better balances the cost estimates of I/O response time and CPU usage. OPTIXOPREF =ON (the default since PK77426) favors index-only access over another index that requires data access when the index has equivalent or better filtering, and the index has equivalent or better support of GROUP BY or ORDER BY order.

When you run the utility on a single partition of an object, RUNSTATS uses the resulting partition-level statistics to update the aggregate statistics for the entire object. RUNSTATS INDEX process all index spaces that are related to a given table space at one time, regardless of list order for optimal processing.

**Attention:** Creating an index gives RUNSTATS the opportunity to collect statistics (by default) that would not have been collected otherwise. Dropping the index, for example, a “unused” index, also drops those statistics. These statistics for one or more columns of an “unused” index may influence the optimizer. This means that dropping an “unused” index may affect the access path for dynamic SQL immediately and static SQL at rebind time. Refer to “Index usage tracking” on page 125 for more information.

When using LISTDEF, RUNSTATS must know the object type that is to be processed before processing can begin. It requires that you specify an object type in addition to the LIST keyword (for example, RUNSTATS INDEX LIST).

**Tip:** Because using the wrong RUNSTATS parameters can be expensive, STATISTICS ADVISOR (which is part of VISUAL EXPLAIN, OPTIMIZATION SERVICE CENTER) should be used to determine what statistics should be gathered for specific SQL statements and recommend the statistics and parameters needed to optimize the SQL submitted for its analysis. For details about STATISTICS ADVISOR, refer to 9.1, “Statistics Advisor”, in *IBM DB2 9 for z/OS: New Tools for Query Optimization*, SG24-74211.

For space information, real-time statistics (RTS) could be used instead of running RUNSTATS. When you run the REFRESH TABLE statement, the only statistic that DB2 updates for the MQT is the cardinality statistic (CARDF).

## 11.2.1 DSTATS

This function calculates the distribution statistics for non-indexed columns to RUNSTATS. The relevant catalog tables are updated with the specified number of highest frequencies and, optionally, with the specified number of lowest frequencies. The new functionality also optionally collects multi-column cardinality for non-indexed column groups and updates the catalog.

RUNSTATS has been enhanced by the distribution statistics (DSTATS) to gather the following additional statistics, allowing the optimizer to do a better job:

- ▶ Frequency value distributions for non-indexed columns or groups of columns
- ▶ Cardinality values for groups of non-indexed columns
- ▶ LEAST frequently occurring values, along with MOST for both index and non-indexed column distributions

Figure 11-2 on page 295 shows the differences between KEYCARD and DSTATS.

### KEYCARD versus Distribution Statistics from an index

State	City	Zipcode
CA	San Jose	95123
CA	San Jose	95110
CA	San Jose	95141
CA	San Jose	95141
CA	Riverside	92504
CA	Riverside	92504
CA	Glendora	91741
TX	Austin	78732

- KEYCARD collects all of the distinct values in all of the 1 to  $n$  key column combinations
- So these are cardinality statistics across column sets... if we had a 3-column index on State, City, Zipcode:

Numcolumns	Card
1	2
2	4
3	6

- FREQVAL NUMCOLS 3 collects

Colvalue	Frequency
CA, San Jose, 95123	1/8 = 0.125
CA, San Jose, 95110	1/8 = 0.125
CA, San Jose, 95141	2/8 = 0.25
CA, Riverside, 92504	2/8 = 0.25
CA, Glendora, 91741	1/8 = 0.125
TX, Austin, 78732	1/8 = 0.125

1

Figure 11-2 KEYCARD versus distribution statistics

These additional statistics can only be gathered by the “stand-alone” RUNSTATS utility. They cannot be gathered when collecting statistics as part of another utility's execution, that is, so-called inline statistics. To enable the collection of cardinality and distribution statistics on any table column, we introduce a new colgroup-spec block. New keywords, COLGROUP, LEAST, MOST, and BOTH, are introduced in this block. The previous limit of 10 names in the COLUMN parameter has been removed. In addition, the existing keywords FREQVAL and COUNT can also be used. Cardinality and distribution statistics are collected only on the columns explicitly specified. Cardinality and distribution statistics are not collected if you specify COLUMN ALL. If frequency statistics do not exist, DB2 assumes that the data is uniformly distributed.

The cardinality statistics are collected in SYSCOLDIST, if the table space is partitioned also in SYSCOLDISTSTATS, and the cardinality statistics of individual columns are collected in the SYSCOLUMNS catalog tables.

CPU and elapsed time may increase for the RUNSTATS utility when you use these functions, depending on the number of COLGROUPS and number of columns specified in the COLGROUP keyword and the additional sortwork that has to be done. Also, RUNSTATS performs a subtask that sorts the column group and partition-level frequency data.

## 11.2.2 HISTORY

History statistics allows the user to keep track of statistics changes over time, without the concern that the main optimizer statistics change as well and can result in access path changes, especially for dynamic SQL. The statistics are in the following DB2 history tables:

- ▶ SYSIBM.SYSCOLDIST\_HIST
- ▶ SYSIBM.SYSCOLUMNS\_HIST
- ▶ SYSIBM.SYSINDEXES\_HIST
- ▶ SYSIBM.SYSINDEXPART\_HIST
- ▶ SYSIBM.SYSINDEXSTATS\_HIST
- ▶ SYSIBM.SYSLOBSTATS\_HIST
- ▶ SYSIBM.SYSTABLEPART\_HIST
- ▶ SYSIBM.SYSTABSTATS\_HIST
- ▶ SYSIBM.SYSTABLES\_HIST
- ▶ SYSKEYTARGETS\_HIST
- ▶ SYSKEYTGTDIST\_HIST

History statistics can now be kept without updating current statistics. Since V8, UPDATE NONE HISTORY ALL is allowed and you can monitor statistic changes over time without concern that access paths may change.

The HISTORY statistics tables are not identical, but do contain the same statistics columns. SYSDBASE contains only one row for each object/part, which is updated when collecting statistics. For the history tables in SYSHIST, rows with statistics for each object/part are inserted when HISTORY is specified with the RUNSTATS utility along with a STATTIME value. The history statistics remain until the MODIFY STATISTICS utility removes them; refer to 11.7, “Modify statistics” on page 313.

The following tables contain only SPACE statistics:

- ▶ SYSTABLEPART\_HIST
- ▶ SYSINDEXPART\_HIST
- ▶ SYSLOBSTATS\_HIST

These tables contain only ACCESSPATH statistics:

- ▶ SYSCOLDIST\_HIST
- ▶ SYSCOLUMNS\_HIST
- ▶ SYSINDEXSTATS\_HIST
- ▶ SYSTABSTATS\_HIST

These tables contain both SPACE and ACCESSPATH related statistics:

- ▶ SYSTABLES\_HIST
- ▶ SYSINDEXES\_HIST

The tables are populated by using the HISTORY option of the RUNSTATS utility; refer to 11.2.2, “HISTORY” on page 296.

Table 11-1 illustrates which tables are updated when the HISTORY option is used.

Table 11-1 Statistic tables updated by the HISTORY option

	SPACE		ACCESSPATH		ALL		LOB table space	
	Table (ALL)	Index (ALL)	Table (ALL)	Index (ALL)	Table (ALL)	Index (ALL)	No param.	Index (ALL)
SYSCOLDIST_HIST				X		X		
SYSCOLUMNS_HIST			X	X	X	X		X
SYSINDEXES_HIST		X		X		X		X
SYSINDEXPART_HIST		X				X		X
SYSINDEXSTATS_HIST				X		X		
SYSTABLEPART_HIST	X	X			X	X	X	X
SYSTABSTATS_HIST			X	X	X	X		
SYSTABLES_HIST	X	X	X	X	X	X	X	X
SYSLOBSTATS_HIST	NP	NP	NP	NP	NP	X	X	X
NP - Not Permitted								

There are also fields added to the “current” statistics fields in DSNDB06, which can be used in determining when utilities are to be executed. Two of the new fields, DSNUM and EXTENTS, are shown in the Example 11-2.

Example 11-2 Statistics for SPACE and EXTENTS

TSNAME	DBNAME	SPACE	SPACEF	DSNUM	EXTENTS
*	*	*	*	*	*
-----	-----	-----	-----	-----	-----
TSPSUPP1	U7G01T11	<b>72000</b>	7.200000000000000E+04	1	<b>7</b>
TSPSUPP	U7G01T11	72000	7.200000000000000E+04	1	<b>7</b>

A listing of the data sets underlying the table space shows the extents, as shown in Example 11-3.

Example 11-3 Data set listing

Command - Enter "/" to select action	Tracks	%Used	XT	Device
-----	-----	-----	-----	-----
DB2V910G.DSNDBD.U7G01T11.TSPSUPP.I0001.A003	<b>1500</b>	?	<b>7</b>	3390
DB2V910G.DSNDBD.U7G01T11.TSPSUPP1.I0001.A003	1500	?	<b>7</b>	3390

The SPACE field can also match the listed tracks. DB2 on a DASD MODEL 3390-3 allocates 12x4 K pages per track, resulting in 48 K per track. Therefore, the relationship between SPACE(F) and tracks can be demonstrated as  $SPACE(F)/48 = TRACKS$ . In this example,  $72000/48 = 1500$ .

The options for HISTORY are the same as for UPDATE, and the scope of the UPDATE affects the options allowed for HISTORY. Table 11-2 illustrates the effect of the UPDATE parameter on HISTORY.

Table 11-2 Allowable HISTORY/UPDATE combinations

UPDATE	HISTORY
ALL	ALL, ACCESSPATH, SPACE, NONE
ACCESSPATH	ACCESSPATH, NONE
SPACE	SPACE, NONE
NONE	NONE

Estimating the space requirements for DB2 objects is easier if you collect and maintain a statistical history of those objects. The accuracy of your estimates depends on how current the statistical data is. To estimate disk storage for user data, ensure that the statistics history is current by using the MODIFY STATISTICS utility to delete outdated statistical data from the catalog history tables.

By using the historical statistics, it is easy to use trend analysis to plan capacity, to identify if utilities are being run too often or not often enough, and to identify objects that may benefit from changes to its physical design. Figure 11-3 shows an example of monitoring space growth over time.

### Monitoring space growth

- Consider the following stats over time
  - SYSTABLEPART\_HIST
    - CARDF, SPACEF
  - SYSINDEXPART\_HIST
    - CARDF, SPACEF
- If we can assume constant growth over time...

```
SELECT MAX(CARDF), MIN(CARDF),
((MAX(CARDF)-MIN(CARDF))*100)/MIN(CARDF),
(DAYS(MAX(STATSTIME))-DAYS(MIN(STATSTIME)))
FROM SYSIBM.SYSTABLEPART_HIST
WHERE DBNAME='DB' AND TSNAME='TS';
```

Min and Max Rows  
% growth  
# days for growth

Figure 11-3 Monitoring space growth with RUNSTATS HISTORY

Assuming that the number of rows is constantly increasing so that the highest number is the latest, the query shows the percentage of rows added over specific time period. This could be extrapolated to provide an annual growth figure.



Another example is the monitoring of the effectiveness of a compression dictionary. To determine how often a compression dictionary should be rebuilt, monitor the PAGESAVE value. Determine when the degradation of the dictionary from the original compression is greater than 10%. At this point, a dictionary rebuild should be scheduled by omitting the KEEPDICTIONARY keyword from the REORG. Refer to Figure 11-4 for guidelines about deciding when to rebuild a dictionary.

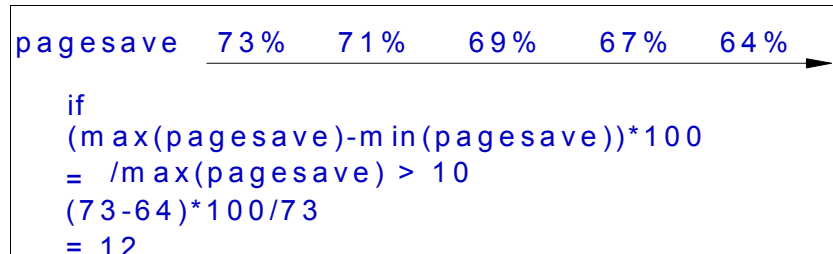


Figure 11-4 Using PAGESAVE to determine dictionary effectiveness

We recommend that the effectiveness of the existing dictionary be monitored by using the column PAGESAVE in the new Version 7 catalog table SYSTABLEPART\_HIST.

We recommend that historical data be collected whenever RUNSTATS is run, as the impact is insignificant.

You need Explicit Database Privilege for the RUNSTATS, CHECK, LOAD, REBUILD INDEX, REORG INDEX, REORG TABLESPACE, and MODIFY STATISTICS utilities, to gather statistics, check indexes and referential constraints for objects in the database, and delete unwanted statistics history records from the corresponding catalog tables.

### 11.2.3 Histograms

Prior to DB2 9, frequency statistics that were collected relied on single values, either from a single column or from multiple columns. The existing optimizer technique uses the column frequency and is good in the evaluation of a small amount of values. Histogram statistics are supported in DB2 9 for z/OS new-function mode to provide the optimizer with more meaningful data. While frequency statistics are beneficial for low cardinality columns, or columns with very few skewed values, they do not help with a large number of skewed values, or when skew occurs across data ranges. When the number of possible values is large, it is detrimental to attempt to collect all skewed values. Collecting more than 100 values can result in increased bind/prepare times. Range frequency or histograms (quantiles) allow you to collect range skew. A histogram is a way of summarizing data measured on an interval scale, which is useful for skewed distributions or distributions with gaps. This technique is not available in RTS.

**Tip:** If you see a big gap, the gap could be a good choice to split. Gaps are best to have “between” quantiles, not “within”.

Histogram statistics collect the frequency statistics of the distinct values of column cardinality over an entire range. This method results in better selectivity and has the potential for better access path selection. As a result, the RUNSTATS utility (RUNSTATS TABLESPACE / RUNSTATS INDEX) now collects information by quantiles. You can specify how many quantiles DB2 uses, from 1 to 100 per column.

Figure 11-5 show the new RUNSTATS Histogram Statistics.

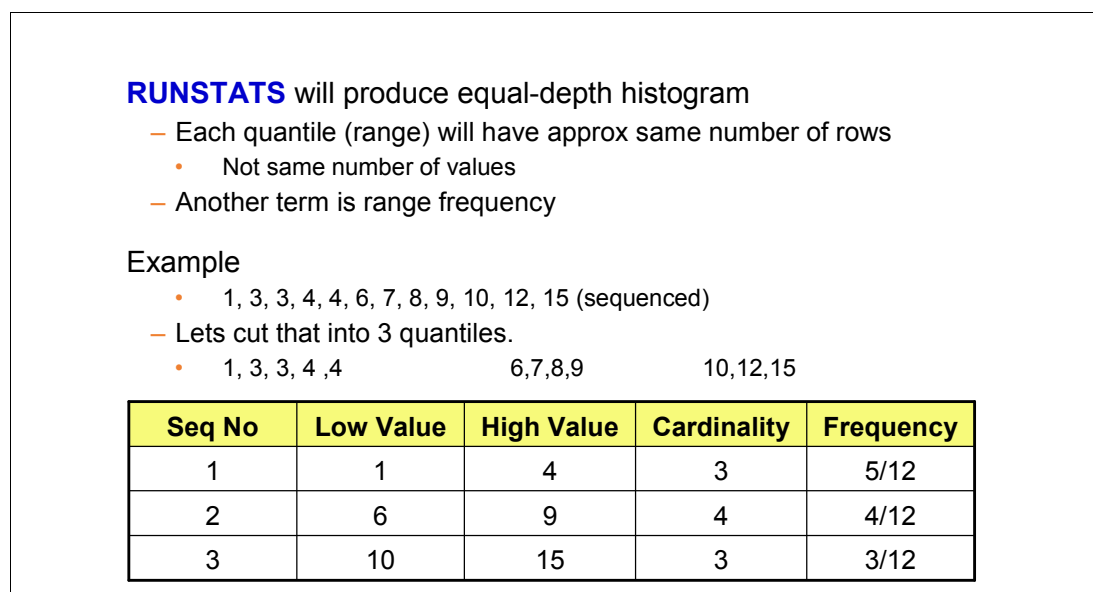


Figure 11-5 RUNSTATS Histogram Statistics

Three new columns have been added to the SYSIBM.SYSCOLDIST and SYSIBM.SYSKEYTGTDIST (as well as SYSCOLDIST\_HIST, SYSCOLDISTSTATS, SYSKEYTGTDIST\_HIST, and SYSKEYGTDISTSTATS) catalog tables:

- ▶ QUANTILENO
- ▶ LOWVALUE
- ▶ HIGHVALUE

There is a column TYPE. When the TYPE is H, the row represent the histogram.

Because the histogram describes the data distribution over the entire range, predicate selectivity is calculated more accurately if the range matches the boundary of any one quantile or any group of consecutive quantiles. Even if there is no perfect match, predicate selectivity interpolation is now done within one or two particular quantiles. With the interpolation done in much smaller granularity, predicate selectivity is expected to be evaluated with more accuracy.

## 11.3 When to run RUNSTATS

RUNSTATS should be run whenever there has been a significant change in the makeup of the data within the table space. This enables the optimizer to accurately select the best access method for the query. We recommend collecting statistics after the following events:

- ▶ After a table is loaded.
- ▶ After an index is physically created.
- ▶ After a table space is reorganized if inline statistics were not collected.
- ▶ After running extensive updates, deletions, or insertions in a table space.
- ▶ After you have run RECOVER TABLESPACE, REBUILD INDEX, or REORG INDEX if inline statistics were not collected.

- ▶ Before running REORG with the OFFPOSLIMIT, INDREFLIMIT, or LEAFDISTLIMIT options.
- ▶ Run RUNSTATS UPDATE SPACE before the REORG/ REBUILD so that DB2 calculates a more accurate estimate for sort work space.
- ▶ After running the ALTER TABLE ROTATE PARTITION statement run RUNSTATS with REORG.
- ▶ To estimate pages changed since RUNSTATS was last run.
- ▶ After any other significant event affecting the data within the table.
- ▶ After the completion of a restarted LOAD, REBUILD INDEX, or REORG job with the STATISTICS option. When you restart these jobs, DB2 does not collect inline statistics. The exception is REORG UNLOAD PAUSE, which, when restarted after the pause, collects statistics. If you have loaded many records, run RUNSTATS SHRLEVEL CHANGE UPDATE SPACE and then a conditional REORG.
- ▶ If you redefine index/table controlled partitions and have NPIs. Use the REORG utility with inline statistics on the partitions that are affected by the change in key range. Use the RUNSTATS utility on the non-partitioned indexes.
- ▶ If you run REORG PART and non partitioning indexes exist, subsequently run RUNSTATS for each non-partitioning index.
- ▶ When columns data or ranges are constantly changing (for example, dates and timestamps). These types of columns can result in old values in the HIGH2KEY and LOW2KEY columns in the catalog. Collect periodically column statistics on these changing columns. Values in HIGH2KEY and LOW2KEY that accurately reflect the true range of data and range predicates will provide more accurate filter factors.
- ▶ Ensure RUNSTATS is executed prior to REBIND in DB2 Version 9. The CLUSTERRATIOF algorithm has changed in DB2 9. There also is a new statistic DATAREPEATFACTOR in DB2 Version 9. These statistic improvements (which are the default behavior with STATCLUS = ENHANCED) enable DB2 to more accurately recognize the sequential nature and density of data when accessed through an index. You will only feel the full effects of DB2 9 optimization on REBINDs after these statistics have been collected and are available for the optimizer.

- STATSINSERTS, STATSDELETES, and STATSUPDATES contain the number of rows inserted, deleted, and updated since the last time a RUNSTATS utility was executed. You can use these statistics to assess the frequency of changes for a particular object and the need for refreshing catalog statistics. Typically, if a sum of these counters is larger than 20% of the number of rows in the table space, a RUNSTATS utility should be executed, as shown in Example 11-4.

*Example 11-4 A sample SQL to evaluate the frequency of changes*

---

```
SELECT DBNAME, NAME, PARTITION
FROM SYSIBM.TABLESPACESTATS
WHERE STATSLASTTIME IS NULL
OR ((STATSINSERTS+STATSDELETES+STATSUPDATES)/TOTALROWS> 0.2
AND STATSINSERTS+STATSDELETES+STATSUPDATES > 30)
OR STATSMASSDELETE > 0

SELECT DBNAME, NAME, PARTITION
FROM SYSIBM.INDEXSPACESTATS
WHERE STATSLASTTIME IS NULL
OR ((STATSINSERTS+STATSDELETES)/TOTALENTRIES > 0.2
AND STATSINSERTS+STATSDELETES > 30)
OR STATSMASSDELETE > 0
```

---

By collecting inline statistics, the total elapsed time of running the two utilities, for example, REORG followed by RUNSTATS, is greatly reduced due to the elimination of the I/O required to perform RUNSTATS. This is done at the cost of a slight increase in CPU time. We highly recommend that inline statistics be collected where a utility allows the option. For more details about inline statistics, refer to 4.2, “Inline executions” on page 76.

You can trigger the RUNSTATS utility periodically or trigger it after percentage of pages changes since RUNSTATS was last run. You can use the DB2 SYSPROC.ADMIN\_UTL\_SCHEDULE, DSNACCOX, or DSNACCOR stored procedures or the IBM DB2 Automation Tool.

If you perform a LOAD PART operation, you should run RUNSTATS INDEX on the non-partitioned secondary indexes to update the catalog data with information about these indexes.

For LOAD operations on a base table that contains an XML column, DB2 does not collect inline statistics for the related XML table space or its indexes.

**Tip:** Run RUNSTATS only on the columns or column groups that might be used as search conditions in a WHERE clause of queries. Use the COLGROUP option to identify the column groups. Collecting additional statistics on groups of columns that are used as predicates improves the accuracy of the filter factor estimate and leads to improved query performance. Collecting statistics on all columns of a table is costly and might not be necessary. Several tools help with RUNSTATS advisory functions.

## 11.4 RUNSTATS options

To ensure that information in the catalog is current and you gather the right statistics, the RUNSTATS utility provides you with many options, which are discussed in this section.

### 11.4.1 REPORT

Indicates whether a set of messages to report the collected statistics is to be generated.

- NO

Indicates that the set of messages will not be sent as output to SYSPRINT.

- YES

Indicates that the set of messages will be sent as output to SYSPRINT. The generated messages depend on the combination of keywords (such as TABLESPACE, INDEX, TABLE, and COLUMN) that you specify with the RUNSTATS utility. However, these messages do not depend on the specification of the UPDATE option. REPORT YES always generates a report of SPACE and ACCESSPATH statistics. KEYCARD specifies that all of the distinct values in all of the 1 to n key column combinations for the specified indexes are to be collected. n is the number of columns in the index.

### 11.4.2 LIST

RUNSTATS supports the use of the LISTDEF to pass a list of objects to a utility statement for further processing. The list generated by the LISTDEF must only contain objects that can be used in the particular RUNSTATS command. Therefore, the LISTDEF for RUNSTATS TABLESPACE must only contain table spaces, and, for RUNSTATS INDEX, only indexes are allowed. If this is not the case, the utility finishes with a return code of 8 and states that an object was invalid for the utility.

If you specify LIST, you cannot specify the PART option. Instead, use the PARTLEVEL option on the LISTDEF statement. RUNSTATS TABLESPACE is invoked once for each item in the list.

### 11.4.3 COLGROUP

When the keyword COLGROUP is specified, the set of columns specified within the COLGROUP keyword is treated as a group. The cardinality values are collected on the column group. You can specify the COLGROUP keyword multiple times (with different sets of columns) for the same table in your RUNSTATS statement. The COLUMN keyword works the same as in previous releases. The cardinality statistics of individual columns are collected in the SYSCOLUMNS catalog table, as in versions prior to DB2 V8.

The cardinality statistics for the specified group(s) of columns are collected in SYSCOLDIST, and, if the table space is partitioned also in SYSCOLDISTSTATS, catalog tables. RUNSTATS TABLESPACE will ignore COLGROUP when processing XML table spaces and indexes. If you want RUNSTATS to also collect distribution statistics, specify the FREQUAL option with COLGROUP.

**Attention:** Collection of distribution statistics via COLGROUP FREQVAL is more expensive than collecting it via INDEX KEYCARD FREQVAL. For example, when the COLGROUP matches the leading columns of an index, it might be cheaper to collect the distribution statistics via the INDEX. The difference between the COLGROUP processing and the INDEX processing is that the COLGROUP processing uses a SORT and a hashing estimation technique for the collection of distribution statistics, whereas in the case of the indexes, the indexes are ordered and are less costly. Therefore, you need to evaluate the necessity for collecting distribution statistics via COLGROUP and restrict the collection to non-indexed columns.

**Considerations:**

- ▶ The length of the COLGROUP value cannot exceed the maximum length of the COLVALUE column in the SYSIBM.SYSCOLDIST catalog table.
- ▶ You cannot specify COLGROUP during inline STATISTICS. To collect distribution statistics on non-indexed columns, specify the COLGROUP keyword in RUNSTATS TABLESPACE TABLE.

## 11.4.4 KEYCARD

KEYCARD provides valuable information with little processing cost. KEYCARD indicates that RUNSTATS will collect all of the distinct values in all of the 1 to  $n$  key column combinations for the specified indexes, where  $n$  is the number of columns in the index. For example, if KEYCARD is specified for a three column index (columns IXC1, IXC2, and IXC3), then extra statistics are collected showing the combination of IXC1+IXC2. The values of IXC1+IXC2+IXC3 are collected in FULLKEYCARDF and IXC1 statistics are held in FIRSTKEYCARDF.

Example 11-5 shows KEYCARD being used with RUNSTATS.

*Example 11-5 RUNSTATS using KEYCARD*

---

```
//SYSIN DD *  
LISTDEF LISTA1 INCLUDE TABLESPACE U7G01T11.TSLINEI  
RUNSTATS TABLESPACE LIST LISTA1 TABLE(ALL) INDEX(ALL KEYCARD)  
REPORT YES
```

---

Example 11-6 shows the output job.

*Example 11-6 Cardinality statistics across column sets with a three-column index*

---

Col1	Col2	Col3
A	B	C
A	B	D
A	B	E
A	B	E
A	C	A
A	C	A
A	D	A
B	B	B

---

then these stats would be collected:

Col1 cardinality = 2

Col1 and Col2 cardinality = 4

Col 1, Col2, and Col3 cardinality = 6

---

We recommend that KEYCARD be used to collect index statistics when there is correlation in a multi-column index keys and queries have less than fully matching equal predicates.

## 11.4.5 FREQVAL

Controls, when specified with the COLGROUP or INDEX option, collect frequent-value statistics. When specified with the COLGROUP option, that frequency statistics are also gathered for the specified group of columns. One group of statistics is gathered for each column. If you specify FREQVAL with INDEX, this keyword must be followed by the NUMCOLS and COUNT keywords. RUNSTATS INDEX will ignore FREQVAL MOST/LEAST/BOTH when processing XML NODEID or VALUES indexes.

The FREQVAL options are:

► **NUMCOLS**

Indicates the number of columns in the index for which RUNSTATS is to collect frequently occurring values. Integer can be a number between 1 and the number of indexed columns. If you specify a number greater than the number of indexed columns, RUNSTATS uses the number of columns in the index. For example, suppose that you have an index defined on three columns: A, B, and C. If you specify NUMCOLS 1, DB2 collects frequently occurring values for column A. If you specify NUMCOLS 2, DB2 collects frequently occurring values for the column set A and B. If you specify NUMCOLS 3, DB2 collects frequently occurring values for the column set A, B, and C. The default value is 1, which means that RUNSTATS collects frequently occurring values on the first key column of the index.

► **COUNT**

Indicates the number of frequently occurring values to be collected from the specified column group. For example, COUNT 20 means that DB2 collects 20 frequently occurring values from the column group. You must specify a value for integer; no default value is assumed. Be careful when specifying a high value for COUNT. Specifying a value of 1000 or more can increase the preparation time for some SQL statements. This option also indicates the number of frequent values that are to be collected. If you specify 15, the utility collects 15 frequent values from the specified key. You must specify COUNT integer with COLGROUP FREQVAL.

► **MOST**

Indicates that the utility will collect the most frequently occurring values for the specified set of columns when COLGROUP is specified or for the specified set of key columns when FREQVAL NUMCOLS COUNT MOST keywords are specified.

► **BOTH**

Indicates that the utility is to collect the most and the least frequently occurring values for the specified set of columns when COLGROUP is specified.

► **LEAST**

Indicates that the utility is to collect the least frequently occurring values for the specified set of columns when COLGROUP is specified.

Example 11-7 shows the collection of distribution statistics for specific columns in a table space and retrieving the most and least frequently occurring values. Collect statistics for the columns EMPLEVEL, EMPGRADE, and EMPSALARY and use the FREQVAL and COUNT keywords to collect the 10 most frequently occurring values for each column and the 10 least frequently occurring values for each column.

*Example 11-7 Collect distribution statistics*

---

```
RUNSTATS TABLESPACE DSN8D81A.DSN8S81E
          TABLE(DSN8810.DEPT)
          COLGROUP(EMPLEVEL,EMPGRADE,EMPSALARY)
          FREQVAL COUNT 10 BOTH
```

---

We recommend starting with the default COUNT of 10 and adjust upward, as required. We do not recommend using a high value for COUNT in an attempt to collect statistics for 100% of the data; this is not necessary and will increase CPU consumption.

## 11.4.6 SAMPLE

SAMPLE indicates the percentage of rows to sample when collecting column statistics. Any value from 1 through 100 can be specified. The default is 25, if SAMPLE is specified. If it is not specified, then sampling is not used, which is equivalent to SAMPLE 100.

When sampling, DB2 still reads all the pages because sampling is at the row level. This enhancement was introduced to minimize the vast amount of CPU required when hashing column values to estimate cardinality.

The default value of 25 provides a saving on CPU without compromising access path selection. In tests, smaller sampling values were also used without noticeable degradation of query performance.

Example 11-8 on page 307, Example 11-9 on page 308, and Example 11-10 on page 309 show the difference in cardinality when RUNSTATS are taken using SAMPLE 10, 25, and 100, respectively. It can be seen from the figures that the difference in the cardinality is marginal in all cases.



Example 11-8 RUNSTATS SAMPLE 10

Se1	Name	Owner	T DB Name	TS Name	Cols	Rows	Checks
*	*	*	* *	*	*	*	*
-----							
*	LINEITEM	PAOLOR4	T U7G01T11	TSLINEI	16	1951548	0
Columns of table							
Select	Column Name	Col No	Col Type	Length	Scale	Null	Def FP
*	*	* *	*	*	*	* *	* *
-----							
	L_ORDERKEY	1	INTEGER	4	0	N	N
	L_PARTKEY	2	INTEGER	4	0	N	N
	L_SUPPKEY	3	INTEGER	4	0	N	N
	L_LINENUMBER	4	INTEGER	4	0	N	N
	L_QUANTITY	5	INTEGER	4	0	N	N
	L_EXTENDEDPRICE	6	FLOAT	4	0	N	N
	L_DISCOUNT	7	FLOAT	4	0	N	N
	L_TAX	8	FLOAT	4	0	N	N
	L_RETURNFLAG	9	CHAR	1	0	N	N
	L_LINESTATUS	10	CHAR	1	0	N	N
	L_SHIPDATE	11	DATE	4	0	N	N
	L_COMMITDATE	12	DATE	4	0	N	N
	L_RECEIPTDATE	13	DATE	4	0	N	N
	L_SHIPINSTRUCT	14	CHAR	25	0	N	N
	L_SHIPMODE	15	CHAR	10	0	N	N
Columns of index							
Se1	Column Name	Seq No	0 Col Type	Length	Scale	Null	Def FP
*	*	* * *	*	*	*	* *	* *
-----							
	L_ORDERKEY	1	A INTEGER	4	0	N	N
	L_SHIPDATE	2	A DATE	4	0	N	N
	L_RETURNFLAG	3	A CHAR	1	0	N	N
	L_SUPPKEY	4	A INTEGER	4	0	N	N
	L_EXTENDEDPRICE	5	A FLOAT	4	0	N	N
	L_DISCOUNT	6	A FLOAT	4	0	N	N
-----							

Example 11-9 RUNSTATS SAMPLE 25

Select	Column Name	Col No	Col Type	Length	Scale	Null	Def	FP	Col Card
*		* *		*		* *	*	*	*
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----
	L_ORDERKEY	1	INTEGER	4		0 N	N	N	487775
	L_PARTKEY	2	INTEGER	4		0 N	N	N	<b>208500</b>
	L_SUPPKEY	3	INTEGER	4		0 N	N	N	10112
	L_LINENUMBER	4	INTEGER	4		0 N	N	N	7
	L_QUANTITY	5	INTEGER	4		0 N	N	N	50
	L_EXTENDEDPRICE	6	FLOAT	4		0 N	N	N	<b>863540</b>
	L_DISCOUNT	7	FLOAT	4		0 N	N	N	11
	L_TAX	8	FLOAT	4		0 N	N	N	9
	L_RETURNFLAG	9	CHAR	1		0 N	N	N	3
	L_LINESTATUS	10	CHAR	1		0 N	N	N	2
	L_SHIPDATE	11	DATE	4		0 N	N	N	2304
	L_COMMITDATE	12	DATE	4		0 N	N	N	2240
	L_RECEIPTDATE	13	DATE	4		0 N	N	N	<b>2400</b>
	L_SHIPINSTRUCT	14	CHAR	25		0 N	N	N	4
	L_SHIPMODE	15	CHAR	10		0 N	N	N	7

Columns of index

Seq	Column Name	No	0	Col Type	Length	Scale	Null	Def	FP	Col Card
*					*		* *	*	*	*
---	---	---	---	---	---	---	---	---	---	---
	L_ORDERKEY	1	A	INTEGER	4		0 N	N	N	487775
	L_SHIPDATE	2	A	DATE	4		0 N	N	N	2304
	L_RETURNFLAG	3	A	CHAR	1		0 N	N	N	3
	L_SUPPKEY	4	A	INTEGER	4		0 N	N	N	10112
	L_EXTENDEDPRICE	5	A	FLOAT	4		0 N	N	N	<b>863540</b>
	L_DISCOUNT	6	A	FLOAT	4		0 N	N	N	11

Example 11-10 RUNSTATS SAMPLE 100

Select	Column Name	Col No	Col Type	Length	Scale	Null	Def	FP	Col Card
*		* *		*		* *	*	*	*
	L_ORDERKEY	1	INTEGER	4		0 N	N	N	487775
	L_PARTKEY	2	INTEGER	4		0 N	N	N	200704
	L_SUPPKEY	3	INTEGER	4		0 N	N	N	10112
	L_LINENUMBER	4	INTEGER	4		0 N	N	N	7
	L_QUANTITY	5	INTEGER	4		0 N	N	N	50
	L_EXTENDEDPRICE	6	FLOAT	4		0 N	N	N	<b>835584</b>
	L_DISCOUNT	7	FLOAT	4		0 N	N	N	11
	L_TAX	8	FLOAT	4		0 N	N	N	9
	L_RETURNFLAG	9	CHAR	1		0 N	N	N	3
	L_LINestatus	10	CHAR	1		0 N	N	N	2
	L_SHIPDATE	11	DATE	4		0 N	N	N	2304
	L_COMMITDATE	12	DATE	4		0 N	N	N	2240
	L_RECEIPTDATE	13	DATE	4		0 N	N	N	2400
	L_SHIPINSTRUCT	14	CHAR	25		0 N	N	N	4
	L_SHIPMODE	15	CHAR	10		0 N	N	N	7

Columns of index

Seq	Column Name	No	Col Type	Length	Scale	Null	Def	FP	Col Card
*		* *		*		* *	*	*	*
	L_ORDERKEY	1	A INTEGER	4		0 N	N	N	487775
	L_SHIPDATE	2	A DATE	4		0 N	N	N	2304
	L_RETURNFLAG	3	A CHAR	1		0 N	N	N	3
	L_SUPPKEY	4	A INTEGER	4		0 N	N	N	10112
	L_EXTENDEDPRICE	5	A FLOAT	4		0 N	N	N	<b>835584</b>
	L_DISCOUNT	6	A FLOAT	4		0 N	N	N	11

Table 11-3 shows a comparison of the performance of various levels of sampling.

Table 11-3 Sampling performance

	RUNSTATS utility		DELTA (%)		
SAMPLE	CPU time	Elapsed time	CPU time	Elapsed time	Total getpages
10	16.38	22.59			50713
25	18.08	24.77	+ 10.37%	+ 9.65%	50713
100	26.86	35.02	+ 63.98%	+ 55.06%	50690
Note: All times are in seconds.					
Note: The number of getpages are approximately equal.					

We recommend that SAMPLE 25 be used initially and testing be performed to find the break even point that balances the savings in CPU against query performance.

**Notes:**

- ▶ If the sample keyword is not supplied, SAMPLE 100 is the RUNSTATS default.
- ▶ If the sample keyword is supplied, SAMPLE 25 is the default.
- ▶ DB2 extrapolates values, based on the percentage supplied by the SAMPLE parameter
- ▶ If 100% accuracy is required, do not use sampling.

## 11.4.7 HISTOGRAM

Indicates, when specified with the COLGROUP or INDEX option, that histogram statistics will be gathered for the specified group of columns or key columns. When used with the INDEX option, histogram statistics can only be collected on the prefix columns with the same order. Key columns for histogram statistics with a mixed order are not allowed. When RUNSTATS collects histogram statistics for partitioned table spaces, it will aggregate them into SYSCOLDIST.

When RUNSTATS collects histogram statistics for partition table spaces or indexes, it will aggregate them into SYSCOLDIST. HISTOGRAM has the following option:

- ▶ **NUMQUANTILES**

Indicates how many quantiles that the utility will collect. The integer value must be equal to or greater than one. The number of quantiles that you specify should never exceed the total number of distinct values in the column or the column group or key columns. The maximum number of quantiles allowed is 100. When the NUMQUANTILES keyword is omitted, NUMQUANTILES takes a default value of 100. Based on the number of records in the table, the number of quantiles is readjusted down to an optimal number.

**Note:** Inline RUNSTATS for histogram statistics is not supported in the REORG, REBUILD, and LOAD utilities.

RUNSTATS TABLESPACE will ignore HISTOGRAM when processing XML table spaces and indexes.

RUNSTATS INDEX will ignore HISTOGRAM when processing XML NODEID or VALUES indexes.

## 11.4.8 UPDATE

The UPDATE option controls the scope of statistics that are collected by the RUNSTATS utility. The options are:

- ▶ **ALL**

Indicates that all collected statistics are to be updated in the catalog.

- ▶ **ACCESSPATH**

Indicates that only statistics relevant to access paths are to be updated in the catalog.

- ▶ **SPACE**

Indicates that only space statistics relevant to helping DBAs monitor the status of objects are to be updated in the catalog.

- ▶ NONE

Indicates that no catalog tables are to be updated with the collected statistics. Executing RUNSTATS always invalidates the dynamic cache; however, when you specify UPDATE NONE REPORT NO, RUNSTATS invalidates statements in the dynamic statement cache without collecting statistics, updating catalogs tables, or generating reports.

## 11.4.9 HISTORY

Records all catalog table inserts or updates to the catalog history tables. The default is supplied by the value that is specified in STATISTICS HISTORY on panel DSNTIPO. Its options are:

- ▶ ALL

Indicates that all collected statistics are to be updated in the catalog history tables.

- ▶ ACCESSPATH

Indicates that the only catalog history table columns that are to be updated are those that provide statistics that are used for access path selection.

- ▶ SPACE

Indicates that only space-related catalog statistics are to be updated in catalog history tables.

- ▶ NONE

Indicates that catalog history tables are not to be updated with the collected statistics.

By specifying RUNSTATS UPDATE NONE HISTORY ALL, history statistics can now be kept without updating current statistics. This way, not only are all statistics collected (ALL), but only the history tables are updated (UPDATE NONE). RUNSTATS does not collect statistics on a clone table, and Access Path Selection (APS) does not use RUNSTATS statistics when accessing a clone table. This is in contrast to real-time statistics, which keeps statistics for both the base and clone objects.

## 11.4.10 FORCEROLLUP

Specifies whether aggregation or rollup of statistics is to take place when RUNSTATS is executed, even if some parts are empty. This keyword enables the optimizer to select the best access path. Its options are:

- ▶ YES

Indicates that forced aggregation or rollup processing will be done, even though some parts might not contain data.

- ▶ NO

Indicates that aggregation or rollup will be done only if data is available for all parts. If data is not available for all parts, the DSNU623I message is issued if the installation value for STATISTICS ROLLUP on panel DSNTIPO is set to NO.

## 11.4.11 SORTDEV

Specifies the device type that DFSORT uses to dynamically allocate the sort work data sets that are required. *device-type* specifies any device type that is acceptable for the DYNALLOC parameter of the SORT or OPTIONS option of DFSORT. For more information, refer to *DFSORT Application Programming Guide*, SC26-7523.

If you omit SORTDEVT, sort is required, and if you did not provide the DD statements that SORT requires for the temporary data sets, SORTDEVT will default to SYSALLDA and the temporary data sets will be dynamically allocated. If you specify SORTDEVT and omit SORTNUM, no value is passed to DFSORT; DFSORT uses its own default.

### 11.4.12 SORTNUM

If you set the DSNZPARMs UTSORTAL and IGNSORTN to YES, the SORTNUM keyword is not required.

If the DSNZPARMs are set to NO, you may have to calculate the size of the sort work data sets. Depending on the type of statistics that RUNSTATS collects, the utility uses the ST01WKnn data sets, the SORTWK01 data set, both types of data sets, or neither. The ST01WKnn data sets are used when collecting statistics on at least one data-partitioned secondary index. You can find the formulas to calculate the approximate size (in bytes) of the ST01WKnn data set and more detailed information in *DB2 Version 9.1 for z/OS Utility Guide and Reference*, SC18-9855.

This keyword specifies the number of required sort work data sets that DFSORT is to allocate. *integer* is the number of temporary data sets that can range from 2 to 255. You need at least two sort work data sets for each sort. The SORTNUM value applies to each sort invocation in the utility. Remember that sort work data sets consume above the line and below the line virtual storage. If you specify too high a value for SORTNUM, the utility may decrease the degree of parallelism due to virtual storage constraints, possibly down to one, meaning no parallelism.

## 11.5 Inline statistics for discarded rows

If you specify the DISCARDDB and STATISTICS options and a row is found with check constraint errors or conversion errors, the row is not loaded into the table and DB2 does not collect inline statistics on it. However, the LOAD utility collects inline statistics prior to discarding rows that have unique index violations or referential integrity violations. In these cases, if the number of discarded rows is large enough to make the statistics significantly inaccurate, run the RUNSTATS utility separately on the table to gather the most accurate statistics.

## 11.6 Fast cached SQL statement invalidation

If users manually update the statistics in the catalog tables, the related dynamic SQL in the cache needs to be invalidated and the next preparation of the statements will cause the access paths to be reevaluated. This enhancement adds new functionality that allows the use of the command shown in Example 11-11.

*Example 11-11 Command cached SQL statement invalidation*

---

UPDATE NONE and REPORT NO keywords

---

This commands can be used on the same RUNSTATS utility execution. This causes the utility to only invalidate statements in the dynamic statement cache without any data access or computation cost. Any statement in the Dynamic Statement Cache that depends on the affected table space or index space will be removed from the cache. The granularity is at the table space/index level (not the table level).

## 11.7 Modify statistics

Consider deleting all or part of the statistics history catalog rows when they are no longer needed. Deleting outdated information from the statistics history catalog tables can improve performance for processes that access data from those tables. You also make the space in the catalog available. The next RUNSTATS or utility job with inline statistics will update the statistic catalog tables with more relevant statistics.

The MODIFY STATISTICS utility lets you delete some or all statistics history rows for a table space, an index space, or an index in the RUNSTATS history tables. However, you could issue SQL DELETE statements to delete the relevant rows.

Like MODIFY RECOVER, the statistics can be deleted in the following ways:

- Statistics gathered before a specific date (keyword DATE)
- Statistics of a specific age (Keyword AGE)
- All statistics history records related to the specified object from all catalog history tables

The syntax diagram for the MODIFY STATISTICS utility is shown in Figure 11-6.

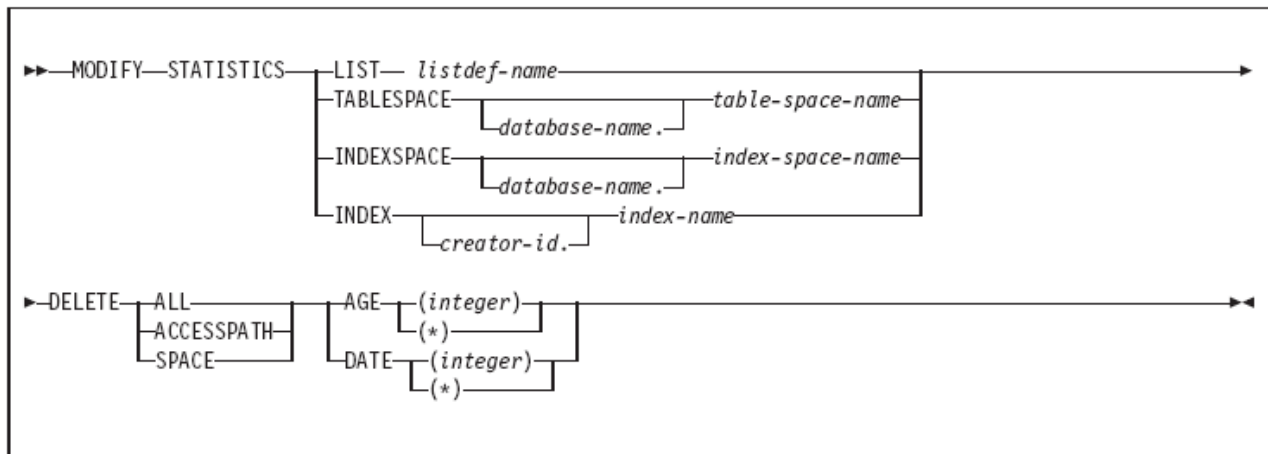


Figure 11-6 MODIFY STATISTICS syntax

As shown in Figure 11-6, you can use LIST to specify the name of a previously defined LISTDEF list name, or TABLESPACE, INDEXSPACE, or INDEX can be explicitly coded.

There are three options to determine which statistics to delete. The delete options are:

- ALL
 

Deletes all statistics history rows that are related to the specified object from all catalog history tables.
- ACCESSPATH
 

Deletes all access-path statistics history rows that are related to the specified object from the following history tables:

  - SYSIBM.SYSCOLDIST\_HIST
  - SYSIBM.SYSCOLUMNS\_HIST
  - SYSKEYTGTDIST\_HIST

► **SPACE**

Deletes all space related statistics history rows related to the specified object from the following history tables:

- SYSIBM.SYSINDEXPART\_HIST
- SYSIBM.SYSTABLEPART\_HIST
- SYSIBM.SYSLOBSTATS\_HIST

Example 11-12 shows an example of deleting SPACE table space statistics based upon a date.

*Example 11-12 MODIFY STATISTICS by date using SPACE*

```
//DSNUPROC.SYSIN DD *
MODIFY STATISTICS TABLESPACE U7G01T11.TSLINEI DELETE SPACE
DATE(20010614)
```

Example 11-13 shows the job output.

*Example 11-13 Output job for table space*

```
DSNU000I DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = TEMP
DSNU050I DSNUGUTC - MODIFY STATISTICS TABLESPACE U7G01T11.TSLINEI DELETE SPACE
DATE(20010614)
DSNU1307I -DB2G DSNUMSTA - 6 SYSIBM.SYSTABLEPART_HIST ROWS WERE DELETED
DSNU1300I -DB2G DSNUMSTA - MODIFY STATISTICS HAS COMPLETED SUCCESSFULLY
DSNU010I DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

Table 11-4 shows the behavior of MODIFY STATISTICS and RUNSTATS SPACE.

*Table 11-4 Deleting statistics gathered by HISTORY SPACE*

	<b>RUNSTATS SPACE</b>		<b>MODIFY Table space</b>	<b>MODIFY Index space</b>	<b>MODIFY Table space</b>	<b>MODIFY Index space</b>
	<b>Table (ALL)</b>	<b>Index (ALL)</b>	<b>Delete Space</b>	<b>Delete Space</b>	<b>Delete (ALL)</b>	<b>Delete (ALL)</b>
SYSCOLDIST_HIST						
SYSCOLUMNS_HIST						
SYSINDEXES_HIST		X		ND		ERR
SYSINDEXPART_HIST		X		X		X
SYSINDEXSTATS_HIST						
SYSTABLEPART_HIST	X	X	X	ND	X	ND
SYSTABSTATS_HIST						
SYSTABLES_HIST	X	X	ND	ND	X	ND
SYSLOBSTATS_HIST	NP	NP	X	NP	X	NP
ND: Not deleted NP: Not permitted						

**Note:** Apply PTF UQ56653 for APAR PQ50666 to eliminate some inconsistencies in deleting RUNSTATS HISTORY entries related to indexes.



When running RUNSTATS TABLESPACE UPDATE SPACE HISTORY SPACE TABLE (ALL) (column 1), DB2 update tables SYSTABLEPART\_HIST and SYSTABLES\_HIST (column 1).

To delete those statistics, use MODIFY STATISTICS TABLESPACE *dbname.tsname* DELETE (ALL) (column 5), because if MODIFY STATISTICS TABLESPACE *dbname.tsname* DELETE SPACE (column 3) is used, only SYSIBM.SYSTABLEPART is deleted, as shown in Example 11-13 on page 314.

If running RUNSTATS TABLESPACE UPDATE SPACE HISTORY SPACE INDEX (ALL) (column 2), DB2 update tables SYSINDEXES\_HIST, SYSIBM.SYSINDEXPART\_HIST, SYSIBM.SYSTABLEPART\_HIST, and SYSTABLES\_HIST (column 2).

To delete those statistics, use MODIFY STATISTICS INDEX *creator.ixname* DELETE (ALL) (column 5) or LISTDEF, as shown in Example 11-12 on page 314, and MODIFY STATISTICS TABLESPACE *dbname.tsname*.

Table 11-5 shows the behavior of MODIFY STATISTICS after RUNSTATS ACCESSPATH completes.

Table 11-5 Deleting statistics gathered by HISTORY ACCESSPATH

	RUNSTATS ACCESSPATH		MODIFY Table space	MODIFY Index space	MODIFY Table space	MODIFY Index space
	Table (ALL)	Index (ALL)	Delete Accesspath	Delete Accesspath	Delete (ALL)	Delete (ALL)
SYSCOLDIST_HIST		X		X		X
SYSCOLUMNS_HIST	X	X	X	ND	X	X
SYSINDEXES_HIST		X		ND		ERR
SYSINDEXPART_HIST						
SYSINDEXSTATS_HIST		X		ND		X
SYSTABLEPART_HIST						
SYSTABSTATS_HIST	X	X	ND	ND	X	ND
SYSTABLES_HIST	X	X	ND	ND	X	ND
SYSLOBSTATS_HIST	NP	NP	NP	NP	X	NP
ND: Not deleted NP: Not permitted						

Table 11-6 shows the behavior of MODIFY STATISTICS after RUNSTATS ALL completes.

Table 11-6 Deleting statistics gathered by HISTORY ALL

	RUNSTATS ALL		MODIFY Table space	MODIFY Index space
	Table (ALL)	Index (ALL)	Delete ALL	Delete ALL
SYSCOLDIST_HIST		X		X
SYSCOLUMNS_HIST	X	X	X	X
SYSINDEXES_HIST		X		ERR
SYSINDEXPART_HIST		X		X
SYSINDEXSTATS_HIST		X		X
SYSTABLEPART_HIST	X	X	X	ND
SYSTABSTATS_HIST	X	X	X	ND
SYSTABLES_HIST	X	X	X	ND
SYSLOBSTATS_HIST	X	X	X	X
ND: Not deleted NP: Not permitted				

You should increase the LOCKMAX parameter for DSNDB06.SYSHIST with ALTER TABLESPACE to avoid timeouts when deleting rows from the history tables.

## 11.8 RUNSTATS and LOB table spaces

The following considerations apply when collecting statistics for LOBs.

### 11.8.1 RUNSTATS on the base table space

In SYSIBM.SYSCOLUMNS, the cardinality of a LOB column, COLCARDF, is determined by counting only non-null and non-zero length columns. HIGH2KEY and LOW2KEY are not applicable for the LOB column and contain blanks.

In SYSIBM.SYSCOLSTATS, the cardinality of a LOB column, COLCARDF, is determined by counting only non-null and non-zero length columns. HIGHKEY, LOWKEY, HIGH2KEY, and LOW2KEY are not applicable for the LOB column and contain blanks.

### 11.8.2 RUNSTATS on the index of the auxiliary table

For an index on the auxiliary table, the CARDF column of SYSIBM.SYSINDEXPART indicates the number of keys in the index that refer to LOBs. Also, LEAFNEAR, LEAFFAR, and PSEUDO\_DEL\_ENTRIES contain meaningful information. The statistics CLUSTERED, CLUSTERRATIO, NEAROFFPOSF, FAROFFPOSF, and LEAFDIST in SYSIBM.SYSINDEXPART are not applicable for the index on the auxiliary table.

### 11.8.3 RUNSTATS on the LOB table space

You can run RUNSTATS on a LOB table space to collect space statistics so that you can determine when the LOB table space should be reorganized. The statistics on the LOB table do not affect access path selection and are not taken into account by the optimizer.

Statistics in SYSIBM.SYSTABLES are not updated except for CARDF and SPACEF, which contain the number of LOBs in the auxiliary table and the number of KB. The statistics in SYSIBM.SYSTABLEPART that are updated are CARDF, SPACEF, PQTY, SQTY, DSNUM, and EXTENTS. PERCACTIVE contains -2 to indicate that this field is not updated for auxiliary tables. Statistics in SYSIBM.SYSCOLUMNS are not updated. HIGH2KEY and LOW2KEY contain blanks for the columns of the auxiliary table, and COLCARDF contains -2 to indicate that this field is not updated for auxiliary tables. The statistics for a LOB table space are stored in a new catalog table called SYSIBM.SYSLOBSTATS. The catalog table SYSIBM.SYSLOBSTATS contains one row for each LOB table space. It holds statistics to manage the space of the LOB table space. It is populated by running RUNSTATS on the LOB table space. The columns of interest are:

- ▶ **FREESPACE:** Kilobytes of free space in extents with respect to high used RBA (HURBA).
- ▶ **AVGSIZE:** Average size of a LOB in bytes.
- ▶ **ORGRATIO:** The percent of organization of the LOB table space. A value of 100 indicates perfect organization (or empty). A value of 1 indicates that the LOB table space is disorganized. A value of 0 means that the LOB table space is totally disorganized.

If the table space that is specified by the TABLESPACE keyword is a LOB table space, you can specify only the following additional keywords:

- ▶ **SHRLEVEL REFERENCE** or **CHANGE**, **REPORT YES** or **NO**
- ▶ **UPDATE ALL** or **NONE**

You cannot specify the **TABLE** option for a LOB table space or options, such as **SAMPLE**, **COLUMN(ALL)**.

When using LISTDEFS to build lists for collecting statistics for LOB table spaces, the following options apply:

- ▶ **ALL**  
Specifies that both related BASE and LOB objects will be included in the list. Auxiliary relationships will be followed from all objects resulting from the initial object lookup and both BASE and LOB objects will remain in the final enumerated list.
- ▶ **BASE**  
Specifies that only base table space (non-LOB) and index spaces are to be included in this element of the list. If the initial search for the object results in a base object, auxiliary relationships are not followed. If the initial search for the object results in a LOB object, the auxiliary relationship is applied to the base table space or index space, and only those objects become part of the resulting list.
- ▶ **LOB**  
Specifies that only LOB table spaces and related index spaces containing indexes on auxiliary tables are to be included in this element of the list. If the initial search for the object results in a LOB object, auxiliary relationships are not followed. If the initial search for the object results in a base object, the auxiliary relationship is applied to the LOB table space or index space, and only those objects become part of the resulting list.

If these options are not specified, then the statistics in the SYSLOBS and SYSLOBS\_HIST will not update. Example 11-14 shows the LISTDEF required to generate the correct lists for RUNSTATS to use when collecting statistics for a LOB table space. Specifying ALL automatically includes all related LOB objects, that is, BASE, AUX, and LOB.

Example 11-14 RUNSTATS with LISTDEF for LOB table spaces

```
//SYSIN DD *
LISTDEF LISTA1 INCLUDE TABLESPACE DSN8D71L.DSN8S71B ALL
RUNSTATS TABLESPACE LIST LISTA1
INDEX(ALL) REPORT YES
UPDATE ALL HISTORY ALL
```

Example 11-15 shows the statements required if not using LISTDEF.

Example 11-15 RUNSTATS for LOB table spaces

```
//SYSIN DD *
RUNSTATS TABLESPACE DSN8D71L.DSN8S71B INDEX (ALL) REPORT YES
UPDATE SPACE HISTORY SPACE
RUNSTATS TABLESPACE DSN8D71L.DSN8S71L INDEX (ALL) REPORT YES
UPDATE ALL HISTORY ALL
RUNSTATS TABLESPACE DSN8D71L.DSN8S71M INDEX (ALL) REPORT YES
UPDATE ALL HISTORY ALL
RUNSTATS TABLESPACE DSN8D71L.DSN8S71N INDEX (ALL) REPORT YES
UPDATE ALL HISTORY ALL
```

We recommend that LISTDEF with option ALL be used for simplicity when building lists for LOB table spaces, because it automatically includes all objects related to the LOB. Also, LOB columns statistics are not used for access path selection. For indexes on auxiliary tables, only the NLEVELS and FIRSTKEYCARDF columns in SYSIBM.SYSINDEXES have an effect on the access path.

Figure 11-7 shows the details collected by RUNSTATS for LOB table spaces.

LOB Management		
<u>SYSLOBSTATS CATALOG STATISTICS</u>		
AVGSIZE	= 4126	(average # bytes per lob)
FREESPACE	= 268	(kilobytes free space in extent)
ORGRATIO	= 1.50	(optimal physical location for active lobs)
<u>SYSTABLEPART Catalog Statistics</u>		
CARD	= 32	(# lobs)
CARDF	= 3.2+01	(# lobs)
SPACE	= 288	(kilobytes of space allocated)
SPACEF	= 2.88E+02	(kb space; V7)
PQTY	= 24	(4K primary extent allocation; V5)
SQTY	= 48	(4K secondary extent allocations; V5)
DSNUM	= 1	(# datasets for tablespace; V7)
EXTENTS	= 2	(total primary & secondary extents; V7)

Figure 11-7 LOB catalog statistics

If using LOBs, the table space holding the actual LOB data is managed by statistics in SYSTABLEPART in the same way as for regular table spaces. However, there are additional statistics saved in SYSLOBSTATS. These are:

- ▶ **AVGSIZE**

This is the average size of all LOBs in the table space, that is, the total number of LOB bytes divided by the number of LOBs.

- ▶ **FREESPACE**

This gives an indication of how many more LOBs can be added in the existing extents already allocated.

- ▶ **ORGRATIO**

This is a measure of fragmentation or non-optimal organization of the LOBs stored in the table space. A value of '100' is optimal. A value of '0' indicates that the LOB table space is totally disorganized.

**Considerations:**

- ▶ The TABLE option is invalid for LOB table space.
- ▶ SHRLEVEL REFERENCE or CHANGE, REPORT YES or NO, and UPDATE ALL or NONE are the only options allowed for LOB table spaces.

As with the other utilities, because a non-partitioned or partitioned base table can contain many auxiliary table space objects (one LOB table space per LOB column and per partition), we recommend that you use a LISTDEF to generate a list of all the objects. Because you cannot specify all keywords for a LOB table space, you probably use two LISTDEFs, one for the base objects and one for the auxiliary objects.

## 11.8.4 RUNSTATS and XML objects

You can use the RUNSTATS utility to gather statistics for XML objects. RUNSTATS TABLESPACE ignores the following keywords for XML table spaces:

- ▶ COLGROUP
- ▶ FREQVAL MOST/LEAST/BOTH
- ▶ HISTOGRAM

RUNSTATS TABLESPACE collects COLCARD for the DOCID column only for an XML table. COLGROUP, FREQVAL MOST/LEAST/BOTH, and HISTOGRAM do not apply, because DB2 is only interested in the number of XML documents stored in the table, not the specific DOCID values.

RUNSTATS INDEX ignores the following keywords for XML indexes or NodeID indexes:

- ▶ KEYCARD
- ▶ FREQVAL MOST/LEAST/BOTH
- ▶ HISTOGRAM

RUNSTATS INDEX collects, for an XML index, HIGH2KEY, LOW2KEY, AVGKEYLEN, FIRSTKEYCARD, FULLKEYCARD, and so on. DB2 is interested in the key value extracted from the XML column by the XPath specified in the XML indexes.





## Verifying data consistency

There are three online utilities, CHECK DATA, CHECK INDEX, and CHECK LOB, which are used to check the consistency of table spaces, index spaces, and LOB table spaces, respectively.

With the CHECK DATA utility, you can:

- ▶ Check one or more table spaces for violations of referential constraints
- ▶ Check one or more table spaces for violations of table check constraints
- ▶ Check the consistency between a base table space and the corresponding LOB or XML table spaces
- ▶ Copy rows that violate a constraint to an exception table
- ▶ Delete rows that violate a constraint
- ▶ Generate REPAIR statements to delete a row that violates a constraint afterwards
- ▶ Set or reset check pending states

With the CHECK INDEX utility, you can:

- ▶ Check if all index entries in the index point to existing data rows
- ▶ Check if all data rows have corresponding entries in the indexes
- ▶ Check one or a list of indexes, and check all indexes of a table space
- ▶ Check indexes on clone tables
- ▶ Check an auxiliary table index to verify that each LOB is represented by an index entry, and that an index entry exists for every LOB

With the CHECK LOB utility, you can:

- ▶ Identify structural defects in a a LOB table space
- ▶ Identify invalid LOBs in a LOB table space
- ▶ Generate REPAIR statements to delete defective or invalid LOBs

There is also an implicit CHECK page option when running COPY and DSN1COPY utility CHECK option.

In this chapter, we focus on new CHECK options introduced with DB2 V8 and DB2 9:

- ▶ Online CHECK or CHECK SHRLEVEL CHANGE
- ▶ The new subsystem parameter UTIL\_TEMP\_STORCLAS
- ▶ Order of executing the different CHECK utilities
- ▶ CHECK DATA of NOT LOGGED table spaces
- ▶ CHECK of LOB and XML data
- ▶ CHECK of clone tables



## 12.1 CHECK SHRLEVEL CHANGE

DB2 V8 introduced CHECK INDEX SHRLEVEL CHANGE. With DB2 9, we also have CHECK DATA and CHECK LOB SHRLEVEL CHANGE. Other utilities that also have SHRLEVEL CHANGE in DB2 9 are REBUILD INDEX and REPAIR LOCATE.

As a reminder, these are the main differences between SHRLEVEL CHANGE and REFERENCE:

- ▶ CHECK SHRLEVEL REFERENCE

This option allows concurrent read access to the data but no write access. The utility executes directly on the data.

- ▶ CHECK SHRLEVEL CHANGE

This option allows concurrent read and write access to the data. The utility executes on a shadow copy of the data and no restrictive states will be set if inconsistencies are detected.

### 12.1.1 Online CHECK INDEX

The CHECK INDEX online utility tests whether indexes are consistent with the data and issues warning messages when it finds an inconsistency. CHECK INDEX is normally run if you suspect inconsistencies exist or want to verify that none exist.

You cannot run CHECK INDEX when the index has a VARBINARY column with the DESC attribute. ALTER the column first to BINARY and REBUILD the index.

DB2 V8 introduced CHECK INDEX SHRLEVEL CHANGE. With the previous SHRLEVEL REFERENCE, applications could read from but could not write to the index, table space, or partition that was to be checked during the entire execution of the CHECK INDEX utility. DB2 unloaded the index entries, sorted the index entries, and scanned the data to validate the index entries. During that time all writers were drained. With SHRLEVEL CHANGE, a different approach was introduced:

- ▶ DB2 first drains all writers and forces the buffers to disk for the specified object and all of its indexes.
- ▶ It then invokes DFSMSDss to copy the specified object and all of its indexes to shadow data sets. This operation can be very fast when FlashCopy V2 is supported on the DASD hardware.
- ▶ DB2 then re-enables read-write access for the specified object and all of its indexes.
- ▶ It then runs the actual CHECK INDEX on the shadow data sets and delete these at TERMUTIL.

If FlashCopy Version 2 is not available, DFSMSDss uses normal DFSMSDss COPY, which might take a long time to copy the objects, and the time during which the data and indexes have read-only access might be bigger than with SHRLEVEL REFERENCE. For this reason, we do not recommend running CHECK SHRLEVEL CHANGE if the page sets involved are not on FlashCopy enabled devices because you will not get the availability you request.

CHECK INDEX with option SHRLEVEL CHANGE increases the availability and satisfies the requirement of customers who cannot afford an application outage. It also dramatically reduces elapsed time due to parallel processing and usage of FlashCopy, as shown in Figure 12-1.

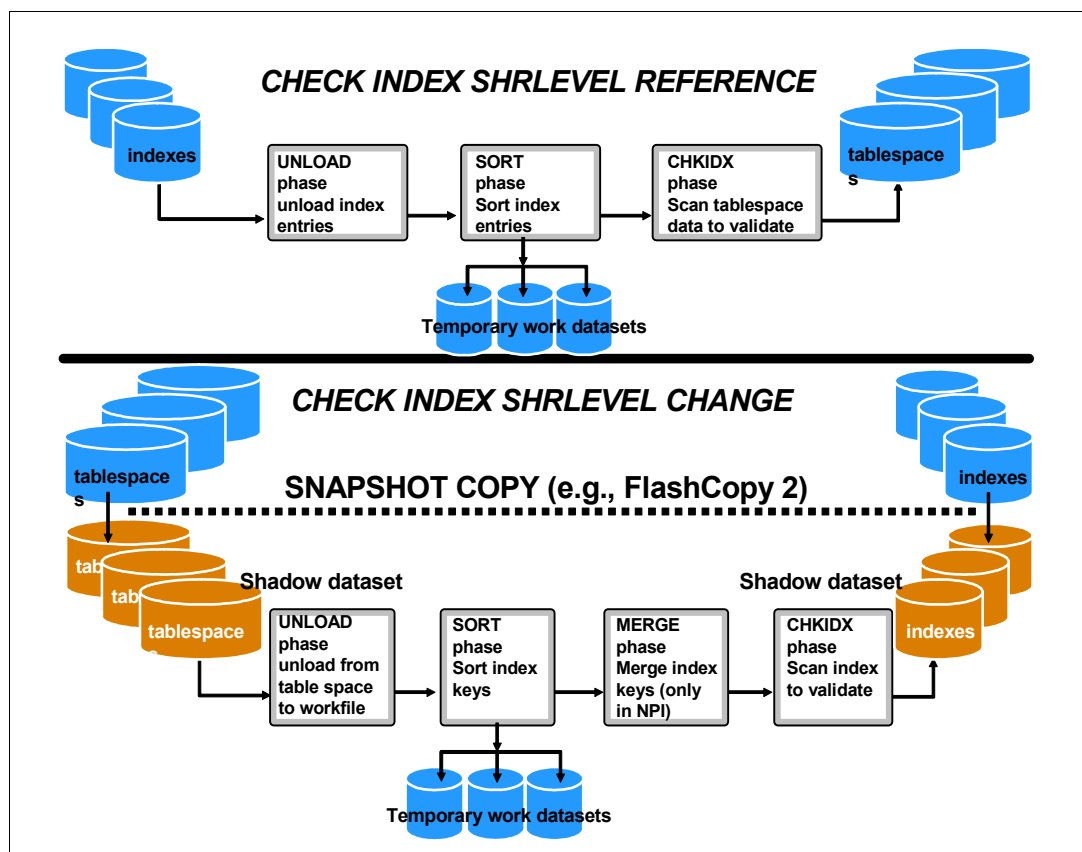


Figure 12-1 Online CHECK INDEX

Example 12-1 shows how to run CHECK INDEX SHRLEVEL CHANGE on all the indexes of a table space. As with other utilities, DRAIN options are used to control the draining operation just before the DFSMSDss COPY function.

*Example 12-1 Invoking CHECK INDEX with SHRLEVEL CHANGE*

```
CHECK INDEX (ALL) TABLESPACE ROEW.ROEW
      SORTDEVT 3390 WORKDDN(TSYSUT1)
      DRAIN_WAIT 20 RETRY 6 RETRY_DELAY 120
      SHRLEVEL CHANGE
```

The resulting job output is shown in Example 12-2 on page 325.

### Example 12-2 Output of CHECK INDEX with SHRLEVEL CHANGE

```
DSNU0001  DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = CHECK
DSNU1044I  DSNUGTIS - PROCESSING SYSIN AS EBCDIC
DSNU0501  DSNUGUTC - CHECK INDEX(ALL) TABLESPACE ROEW.ROEW SORTDEVT 3390 WORKDDN(TSYSUT1) DRAIN_WAIT 20 RETRY 6
RETRY_DELAY 120 SHRLEVEL CHANGE
DSNU421I  DSNUKCPX - START OF DFSMS MESSAGES

                    5695-DF175 DFSMSDSS V1R09.0 DATA SET SERVICES      2009.300 22:26

PARALLEL
ADR101I (R/I)-RI01 (01), TASKID 001 HAS BEEN ASSIGNED TO COMMAND 'PARALLEL'
COPY DATASET(FILTERDD(SYS00001)) DEBUG(FRMSG(DTL))-
REPLACEUNCONDITIONAL RENUNC( -
    (*.*.*.I0001.*.*.*.J0001.*), -
    (*.*.*.J0001.*.*.*.I0001.*)) -
STORCLAS(SCDB2FC ) BYPASSACS(**) -
FCNOCOPY FASTREPLICATION(PREFERRED) -
SHARE CANCELERROR TOLERATE(ENQFAILURE)
ADR101I (R/I)-RI01 (01), TASKID 002 HAS BEEN ASSIGNED TO COMMAND 'COPY '
ADR109I (R/I)-RI01 (01), 2009.300 22:26:35 INITIAL SCAN OF USER CONTROL STATEMENTS COMPLETED
ADR014I (SCH)-DSSU (02), 2009.300 22:26:35 ALL PREVIOUSLY SCHEDULED TASKS COMPLETED. PARALLEL MODE NOW IN EFFECT
ADR050I (002)-PRIME(01), DFSMSDSS INVOKED VIA APPLICATION INTERFACE
ADR016I (002)-PRIME(01), RACF LOGGING OPTION IN EFFECT FOR THIS TASK
ADR006I (002)-STEND(01), 2009.300 22:26:35 EXECUTION BEGINS
ADR711I (002)-NEWDS(01), DATA SET DB20D.DSNDBC.ROEW.ROEW.J0001.A001 HAS BEEN ALLOCATED WITH NEWNAME
DB20D.DSNDBC.ROEW.ROEW.I0001.A001 USING STORCLAS SCDB2FC, DATACLAS DB2TS, AND MGMTCLAS MIGNOBU
ADR806I (002)-TOMI (03), DATA SET DB20D.DSNDBC.ROEW.ROEW.J0001.A001 COPIED USING A FAST REPLICATION FUNCTION
ADR711I (002)-NEWDS(01), DATA SET DB20I.DSNDBC.ROEW.IRROEWR1.J0001.A001 HAS BEEN ALLOCATED WITH NEWNAME
DB20I.DSNDBC.ROEW.IRROEWR1.I0001.A001 USING STORCLAS SCDB2FC, DATACLAS DB2IX, AND MGMTCLAS MIGNOBU
ADR806I (002)-TOMI (03), DATA SET DB20I.DSNDBC.ROEW.IRROEWR1.J0001.A001 COPIED USING A FAST REPLICATION FUNCTION
ADR711I (002)-NEWDS(01), DATA SET DB20I.DSNDBC.ROEW.IRROEWR3.J0001.A001 HAS BEEN ALLOCATED WITH NEWNAME
DB20I.DSNDBC.ROEW.IRROEWR3.I0001.A001 USING STORCLAS SCDB2FC, DATACLAS DB2IX, AND MGMTCLAS MIGNOBU
ADR806I (002)-TOMI (03), DATA SET DB20I.DSNDBC.ROEW.IRROEWR3.J0001.A001 COPIED USING A FAST REPLICATION FUNCTION
ADR711I (002)-NEWDS(01), DATA SET DB20I.DSNDBC.ROEW.IRROEWR4.J0001.A001 HAS BEEN ALLOCATED WITH NEWNAME
DB20I.DSNDBC.ROEW.IRROEWR4.I0001.A001 USING STORCLAS SCDB2FC, DATACLAS DB2IX, AND MGMTCLAS
MIGNOBU
ADR806I (002)-TOMI (03), DATA SET DB20I.DSNDBC.ROEW.IRROEWR4.J0001.A001 COPIED USING A FAST REPLICATION FUNCTION
ADR711I (002)-NEWDS(01), DATA SET DB20I.DSNDBC.ROEW.IRROEWR5.J0001.A001 HAS BEEN ALLOCATED WITH NEWNAME
DB20I.DSNDBC.ROEW.IRROEWR5.I0001.A001 USING STORCLAS SCDB2FC, DATACLAS DB2IX, AND MGMTCLAS MIGNOBU
ADR806I (002)-TOMI (03), DATA SET DB20I.DSNDBC.ROEW.IRROEWR5.J0001.A001 COPIED USING A FAST REPLICATION FUNCTION
ADR801I (002)-DDDS (01), DATA SET FILTERING IS COMPLETE. 5 OF 5 DATA SETS WERE SELECTED: 0 FAILED SERIALIZATION AND 0
FAILED FOR OTHER REASONS.
ADR454I (002)-DDDS (01), THE FOLLOWING DATA SETS WERE SUCCESSFULLY PROCESSED
    DB20D.DSNDBC.ROEW.ROEW.J0001.A001
    DB20I.DSNDBC.ROEW.IRROEWR1.J0001.A001
    DB20I.DSNDBC.ROEW.IRROEWR3.J0001.A001
    DB20I.DSNDBC.ROEW.IRROEWR4.J0001.A001
    DB20I.DSNDBC.ROEW.IRROEWR5.J0001.A001
ADR006I (002)-STEND(02), 2009.300 22:26:36 EXECUTION ENDS
ADR013I (002)-CLTSK(01), 2009.300 22:26:36 TASK COMPLETED WITH RETURN CODE 0000
ADR012I (SCH)-DSSU (01), 2009.300 22:26:36 DFSMSDSS PROCESSING COMPLETE. HIGHEST RETURN CODE IS 0000
DSNU422I  DSNUKCPX - END OF DFSMS MESSAGE
DSNU3340I  DSNUKPIK - UTILITY PERFORMS DYNAMIC ALLOCATION OF SORT DISK SPACE
DSNU1044I  DSNUGTIS - PROCESSING SYSTEMPL AS EBCDIC
DSNU395I  DSNUKPIK - INDEXES WILL BE CHECKED IN PARALLEL, NUMBER OF TASKS = 9
DSNU3342I  DSNUKPIK - NUMBER OF OPTIMAL SORT TASKS = 4, NUMBER OF ACTIVE SORT TASKS = 4
DSNU701I  -DB02 DSNUKIUL - 250832 INDEX ENTRIES UNLOADED FROM 'ROEW.ROEW'
DSNU705I  DSNUKPIK - UNLOAD PHASE COMPLETE - ELAPSED TIME=00:00:02
DSNU719I  DSNUKPIK - 62708 ENTRIES CHECKED FOR INDEX '##T.I_ROEW_1'
DSNU719I  DSNUKPIK - 62708 ENTRIES CHECKED FOR INDEX '##T.I_ROEW_3'
DSNU719I  DSNUKPIK - 62708 ENTRIES CHECKED FOR INDEX '##T.I_ROEW_4'
DSNU719I  DSNUKPIK - 62708 ENTRIES CHECKED FOR INDEX '##T.I_ROEW_5'
DSNU720I  DSNUKPIK - SORTCHK PHASE COMPLETE, ELAPSED TIME=00:00:00
DSNU010I  DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

In this case, we also used the new DB2 subsystem parameter UTIL\_TEMP\_STORCLAS (introduced by APAR PK41711) because our base DB2 VSAM data sets are allocated on PPRC volumes. PPRC means that these volumes are paired for peer to peer remote copy. PPRC volumes are paired with another volume so that a mirror copy can be created either synchronously or asynchronously.

The new UTIL\_TEMP\_STORCLASS removes a restriction with the BACKUP SYSTEM utility where a volume could not be the source and target for a FlashCopy operation at the same time. So while a BACKUP SYSTEM background copy was still going on, you could not create the shadow data sets with data set level FlashCopy on the same volume (then the volume would be a target).

Normally, as with other utilities, the online CHECK utilities allocates the shadow data sets in the same storage class as the base data sets. If these volumes are PPRC volumes, this would slow down the FlashCopy process because of the system processing impact caused by PPRC. Therefore, DB2 does not allow you to use FlashCopy if the temporary shadows are created on a PPRC volume. The subsystem parameter UTIL\_TEMP\_STORCLAS can be used to specify another storage class that would be passed to DFDSS by the CHECK utilities in order to provide a way to have shadow data sets allocated on non-PPRC volumes. This parameter must specify a storage class that points to a storage group with non-PPRC volumes. This storage class is then passed to DFDSS in the COPY command. In Example 12-2 on page 325, the value set in the UTIL\_TEMP\_STORCLAS is SCDB2FC.

If the shadow data sets are created on a PPRC volume, the FlashCopy will fail and normal COPY will be used. For big data sets, this is much slower than FlashCopy and will make the data unavailable for writers for a long time. The FlashCopy will fail with the message shown in Example 12-3.

*Example 12-3 When FlashCopy fails because of shadow data sets allocated on PPRC volumes*

---

```
ADR918I (002)-VDSS (01), FAST REPLICATION COULD NOT BE USED FOR DATA SET
DB2GD.DSNDDBC.ROEW.ROEW.I0001.A001, RETURN CODE 15
```

```
IGD17268I 51 VOLUMES WERE NOT USED FOR FAST RELICATION BECAUSE
OF PPRC PRIM CURRENTLY ACTIVE - ANTRQST QFRVOLS VOLUME RSN(2)
```

---

## 12.1.2 Online CHECK DATA

The CHECK DATA utility checks table spaces for violations of referential and table check constraints, and reports information about violations that it detects. CHECK DATA checks for consistency between a base table space and the corresponding LOB or XML table spaces.

You cannot run CHECK DATA on LOB table spaces, XML table spaces, and on encrypted data. The utility does not check informational referential constraints introduced in DB2 V8.

CHECK DATA is normally run after a conditional restart or a point-in-time recovery on all table spaces where parent and dependent tables might not be synchronized or where base tables and auxiliary tables might not be synchronized.

Prior to Version 9, access to tables spaces was read only when CHECK DATA was run. This means that as long as CHECK DATA was executing you could not modify the data. If you had complex referential sets or very large databases, you likely had issues when running CHECK DATA. If CHECK DATA found a violation, then the table space was placed into the CHECK pending (CHKP) state, which made all the data unavailable even though as little as one row might have an issue. But it was also possible to copy the violating rows to an exception table and delete them, without leaving the table space in the restrictive CHKP status.

In DB2 9 you can run CHECK DATA online (with SHRLEVEL CHANGE). As with CHECK INDEX, an online CHECK DATA will run on a shadow copy of the data and indexes, as shown in Figure 12-2. DB2 makes the shadow copy using DFSMS ADRDSSU. This copy can complete quickly if you have data set level FlashCopy V2 enabled. Even if you do not have FlashCopy, the availability can be improved because the outage is reduced to the time it takes to make the copy instead of the entire utility execution, but you evaluate this scenario to see if this is true for your situation.

Another difference is that with SHRLEVEL CHANGE, because of the shadow copies, when violations are found, DB2 will not set the CHECK pending state and will not delete rows in violation. Instead, CHECK DATA will create a PUNCHDDN containing REPAIR LOCATE DELETE input that can be run against the active data afterwards to delete the rows in violation.

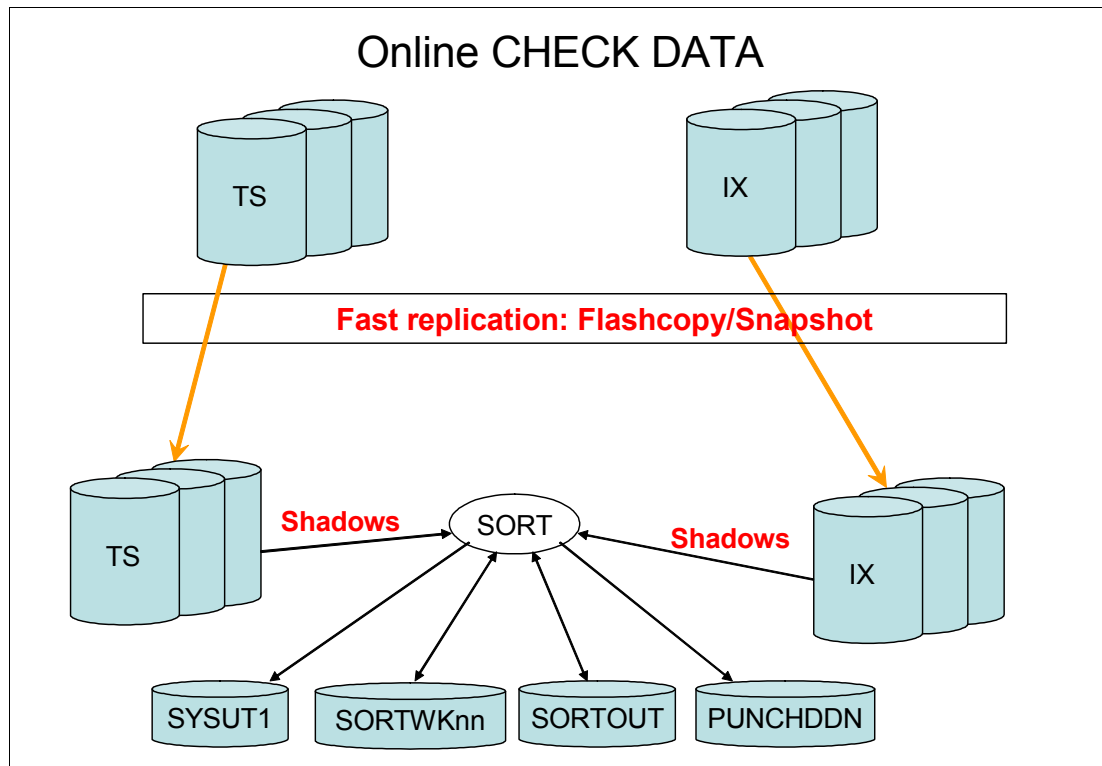


Figure 12-2 Online CHECK DATA

Example 12-4 shows how to run CHECK DATA SHRLEVEL CHANGE on a table space. As with other utilities, DRAIN options are used to control the draining operation just before the FlashCopy. The REPAIR LOCATE DELETE statements, if violations are found, will be written to the PUNCHDDN data set. With SHRLEVEL CHANGE, you cannot specify the DELETE and FOR EXCEPTION keywords. If you are using PPRC volumes, use the subsystem parameter UTIL\_TEMP\_STORCLAS to redirect the shadow data sets to non-PPRC volumes.

---

*Example 12-4 Invoking CHECK DATA with SHRLEVEL CHANGE*

---

```

TEMPLATE TSORTOUT DSN('&US..&SS..&DB..&SN..S&JU(3,5)..#&TI.')
      SPACE TRK MAXPRIME 65536
      DISP(MOD,DELETE,CATLG)
TEMPLATE TSYSUT1  DSN('&US..&SS..&DB..&SN..U&JU(3,5)..#&TI.')
      DISP(MOD,DELETE,CATLG)
TEMPLATE TSYSERR  DSN('&US..&SS..&DB..&SN..E&JU(3,5)..#&TI.')
      DISP(MOD,DELETE,CATLG)
TEMPLATE TSYPUNC  DSN('&US..&SS..&DB..&SN..P&JU(3,5)..#&TI.')
      DISP(MOD,CATLG,CATLG)
CHECK DATA TABLESPACE DB2R601.GLWSEMP
      SCOPE REFONLY
      SORTDEVT 3390
      DRAIN_WAIT 20 RETRY 6 RETRY_DELAY 120
      WORKDDN(TSYSUT1,TSORTOUT) ERRDDN(TSYSERR)
      PUNCHDDN(TSYPUNC)
      SHRLEVEL CHANGE

```

---

### 12.1.3 Online CHECK LOB

The CHECK LOB utility can be used to verify if a LOB table space has defects or invalid LOBs.

When CHECK LOB is run in V8, access is restricted completely for the whole LOB table space for the duration of the utility's run. Due to the table space's inherent size, this could be a significant amount of time. In DB2 9 with CHECK LOB SHRLEVEL REFERENCE and APAR PK55972, writers are drained at the beginning and the LOB table space is only put in check pending (CHKP) status when an error is found, which makes it unavailable afterwards for all applications with SQLCODE- 904, resource unavailable with reason code 00C900A3.

The CHECK LOB utility has also been enhanced to enable online execution. DB2 9 allows SHRLEVEL CHANGE for checking LOBs using a shadow copy of the LOB table space. Just like with CHECK DATA, DB2 uses the DFSMS ADRDSSU utility to create the shadow, as shown in Figure 12-3. Similarly, the check is executed against the shadow and REPAIR statements are generated that you subsequently can run against the actual LOB table space.

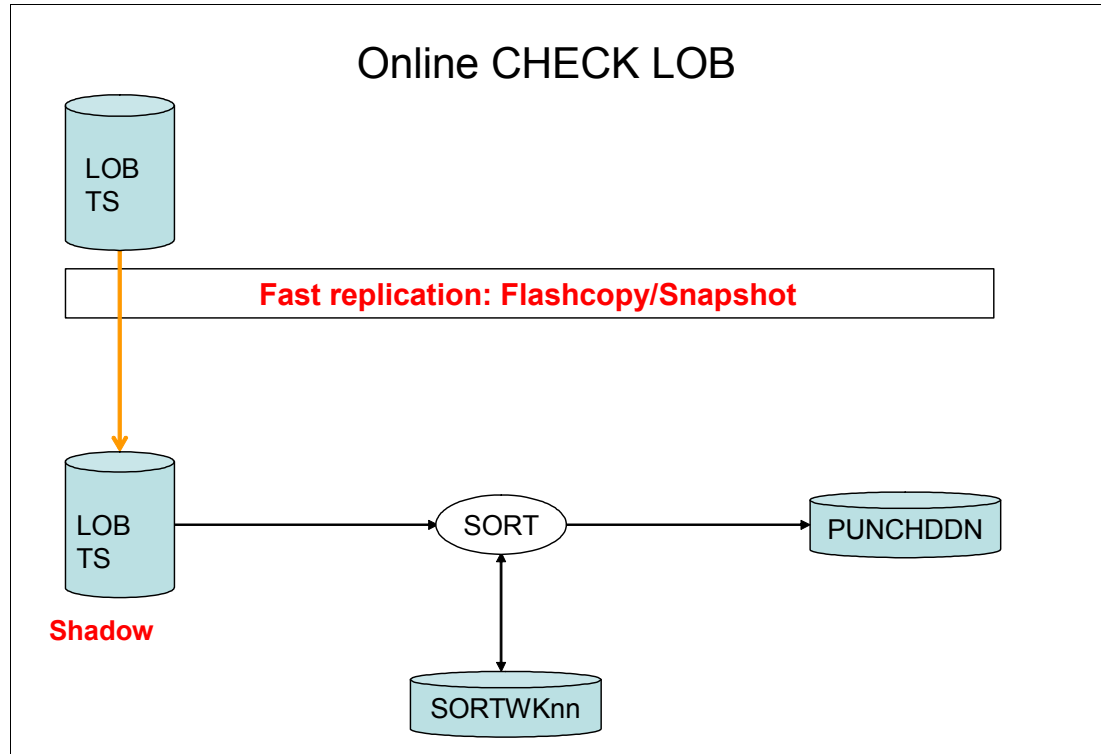


Figure 12-3 Online CHECK LOB

In Example 12-5, we show an example of how to run CHECK LOB SHRLEVEL CHANGE on a LOB table space. As with other utilities, DRAIN options are used to control the draining operation just before the FlashCopy. The REPAIR LOCATE DELETE statements, if violations are found, will be written to the PUNCHDDN data set. If you are using PPRC volumes, use the subsystem parameter UTIL\_TEMP\_STORCLAS to redirect the shadow data sets to non-PPRC volumes.

*Example 12-5 Invoking CHECK LOB with SHRLEVEL CHANGE*

---

```

TEMPLATE TSYPUNC DSN('&US..&SS..&DB..&SN..P&JU(3,5)..#&TI.')
      DISP(MOD,CATLG,CATLG)
CHECK LOB TABLESPACE DSN8D91L.DSN8S91M
      SORTDEVT 3390
      DRAIN_WAIT 20 RETRY 6 RETRY_DELAY 120
      PUNCHDDN(TSYPUNC)
      SHRLEVEL CHANGE
  
```

---

## 12.2 Sequence when executing the CHECK utilities

It is important to understand the sequence to follow when running jobs with different CHECK options. The sequence depends on the following considerations.

CHECK DATA always uses the indexes on the table space involved to access the data. Therefore, you need to run CHECK INDEX before running CHECK DATA.

- ▶ If the table contains no LOB columns:
  - Run CHECK INDEX on the primary key indexes and foreign key indexes to ensure that the indexes that CHECK DATA uses are valid.
  - Run CHECK DATA.
- ▶ If the table contains LOB columns:

The CHECK DATA utility accesses the base table space and all related auxiliary indexes of the LOB table spaces. CHECK DATA relies on information in the LOB table space and the auxiliary indexes being correct. It might be necessary to first run CHECK LOB on the LOB table space and CHECK INDEX and REBUILD INDEX on the auxiliary indexes to ensure the validity of the LOB table spaces and auxiliary indexes. If the LOB table space is in either the check pending (CHKP) status or recover pending (RECP) status, or if the index on the auxiliary table is in rebuild pending (RBPD) status, CHECK DATA issues an error message and fails.

  - a. Run CHECK LOB on the LOB table space to verify the integrity of the LOB table space.
  - b. Run CHECK INDEX on the auxiliary index to verify that each LOB is represented by an index entry, and that an index entry exists for every LOB. Ensure that all LOB column definitions are complete.
  - c. Run CHECK INDEX on the indexes on the base table space.
  - d. Run CHECK DATA.
- ▶ If the table contains XML columns:

Query the SYSIBM.SYSXMLRELS catalog table if you need to determine the XML objects.

  - a. Run CHECK INDEX on the node ID index of each XML column.
  - b. Run CHECK DATA on a XML table space.

## 12.3 CHECK DATA of NOT LOGGED table spaces

As mentioned, with CHECK DATA SHRLEVEL REFERENCE, you can copy violating rows to an exception table and delete them, without leaving the table space in the restrictive CHKP status. This is done by specifying the FOR EXCEPTION keyword with DELETE YES:

- ▶ For a normal LOGGED table spaces, if LOG YES is specified or defaulted, deleted rows are logged. If LOG NO is specified and rows are deleted, the changed table space is put in the *restrictive COPY pending state* and the indexes that were defined with the COPY YES attribute are put in the *informational COPY-pending state*.
- ▶ For NOT LOGGED table spaces, if LOG YES is in effect, CHECK DATA issues the message DSNU1153I to inform the user that the LOG YES option is ignored. If rows are deleted in a NOT LOGGED table space, CHECK DATA sets ICOPY pending regardless of the LOG option.



## 12.4 CHECK of LOB and XML data

There are many similarities between physically storing LOB data and XML data in DB2:

- ▶ The data belonging to a LOB column is stored in an auxiliary table in a separate LOB table space. A ROWID column is used to link the base table with the LOB table space via the auxiliary index.
- ▶ The data belonging to a XML column is stored in a XML table in a separate XML table space. A BIGINT column, called DocID, is used to link the base table with the XML table via the NodeID index. The NodeID index on the XML table allows access to XML documents corresponding to the base row. The NodeID index is made up of the DocID and a node ID. To locate the corresponding XML column, DB2 will use the DocID from the base row and, starting with the minimum node ID, DB2 will use this key to search the NodeID index. The NodeID index may contain multiple entries per XML data row, depending on the nodes grouped in the XML data row. Each DB2 table that contains one or more XML columns will also have a DocID index on the DocID BIGINT column. The DocID index is used when the XML table is accessed first through an XML index to search the XML data. When the data is found, the DocID from the XML table is used to access the base table using the DocID index. The XML indexes can be created on the XML column(s) to improve the efficiency of queries. An XML pattern is used to indicate which parts of the XML column are indexed.

The DB2 utilities handle XML objects similar to the way that they handle LOB objects:

- ▶ CHECK DATA can check the consistency between a base table space and the corresponding XML table spaces.
- ▶ CHECK INDEX can check XML indexes, DocID indexes, and NodeID indexes.
- ▶ There is no CHECK XML utility to check defects in the XML table space.

### 12.4.1 CHECK DATA

The CHECK DATA utility checks for consistency between a base table space and the corresponding LOB or XML table spaces. Run CHECK DATA after a conditional restart or a point-in-time recovery where base tables and LOB/XML table spaces might not be synchronized or when the base table space is in the auxiliary check pending state (ACHKP) or auxiliary warning state (AUXW).

CHECK DATA reports the following errors:

- ▶ Orphan LOBs/XMLs: An orphan LOB/XML value is a LOB/XML entry found in the auxiliary index or NodeID index, but not referenced by the base table space. An orphan can result if you recover the base table space to a point in time prior to the insertion of the base table row or prior to the definition of the LOB/XML column.
- ▶ Missing LOBs/XMLs: A missing LOB/XML value is a LOB/XML referenced in the base table space, but the LOB/XML entry is not in the auxiliary/NodeID index. A missing LOB/XML can result if you recover the LOB/XML table space and auxiliary/NodeID index to a point in time when the LOB/XML value is not in the LOB/XML table space. This could be a point in time prior to the first insertion of the LOB/XML into the base table, or when the LOB/XML column is null or has a zero length.

- **Out-of-sync LOBs:** An out-of-sync LOB error occurs when DB2 detects a LOB that is found in both the base table and the auxiliary index, but the LOB entry in the auxiliary index is at a different version. A LOB column is also out-of-sync if the base table LOB column is null or has a zero length, but the LOB is found through the auxiliary index. An out-of-sync LOB can occur anytime you recover the LOB table space and auxiliary index or the base table space to a prior point in time. There is no versioning of XML documents similar to LOBs. Therefore, no equivalent checks can be done, which means an out-of-sync situation cannot occur.
- **Invalid LOBs/XMLs in the base table space:** An invalid LOB/XML is an uncorrected LOB/XML column error found by a previous execution of CHECK DATA AUXERROR INVALIDATE. An invalid LOB/XML in the base table space has the invalid flag set in the base table space. CHECK DATA does not report invalid LOBs in the LOB table space.

CHECK DATA only inspects the auxiliary/NodeID index and does not access the LOB/XML table space itself. CHECK DATA does not check the DocID index or any XML index. CHECK LOB must be run on the LOB table space to find invalid LOBs in the LOB table space.

You can use the SCOPE AUXONLY to limit the checks for LOBs and XML only; otherwise, the CHECK DATA also checks the referential integrity constraints and table constraints. With DB2 9, the CHECK DATA keyword AUXERROR covers both LOB data and XML data. With SCOPE AUXONLY, you can also use LOBERROR or XMLERROR to distinguish between LOB data and XML data.

Depending on the REPORT or INVALIDATE clause specified, the following actions are performed when using CHECK DATA SHRLEVEL REFERENCE:

- **With AUXERROR/LOBERROR/XMLERROR REPORT:** If the base table space is not yet in auxiliary check pending (ACHKP) status, DB2 drains all SQL writers and sets the base table space and auxiliary table space in UTRO. The data is still available for SQL readers. If errors are found, then the base table space is set to the auxiliary check pending (ACHKP) status. The whole table, or partition, if the table space is partitioned, is now unavailable for applications. The applications receive SQLCODE - 904 resource unavailable reason code 00C900C5. If CHECK DATA encounters no errors and the auxiliary check pending (ACHKP) status was already set, it is reset. If CHECK DATA encounters only invalid LOB or XML columns, the base table space is set to the auxiliary warning status (AUXW), and the table is available for applications. You can use SQL to populate invalid LOBs or XML documents using update or to delete the entire row; however, any other attempt to access the LOB or XML column results in a -904 SQL return code.
- **With AUXERROR/LOBERROR/XMLERROR INVALIDATE:** If the base table space is not yet in auxiliary check pending (ACHKP) status, DB2 drains all SQL readers and writers, sets the base table space and auxiliary table space in UTUT, and the data is no longer available for SQL. DB2 invalidates the LOB and XML columns that are in error (it sets the invalid flag in the base table space for the LOB or XML value). DB2 resets the invalid status of LOB and XML columns that have been corrected. If CHECK DATA encounters no more invalid LOBs or XML documents, then the table space is reset in a no pending state. If invalid LOB or XML columns remain, CHECK DATA sets the base table space to the auxiliary warning (AUXW) status and the table is available for applications. You can use SQL to populate invalid LOBs or XML documents using update or delete the entire row; however, any other attempt to access the column results in a -904 SQL return code. The base table and all other non-invalid LOBs and XML documents are available for applications.

Depending on the REPORT or INVALIDATE clause specified, the following actions are performed when using CHECK DATA SHRLEVEL CHANGE:

- ▶ With AUXERROR REPORT: CHECK DATA runs on the Snapshot™ data sets, and during its execution, the base table space remains available for applications if it was not yet in the ACHKP status. If errors are found, the shadow data sets are deleted, but the base table space is not set to the auxiliary check pending (ACHKP) status at the end of the utility. The whole table stays available for applications. If the base table space was already in ACHKP and no errors were found anymore, the ACHKP status is not reset at the end.
- ▶ With AUXERROR INVALIDATE: CHECK DATA runs on the Snapshot data sets, and during its execution, the base table space remains available for applications if it was not yet in the ACHKP status. DB2 does not invalidate the LOB/XML columns that are in error in the base table space (it does not set the invalid flag in the base table). DB2 does not reset the invalid status of LOB/XML columns that have been corrected in the base table space. However, CHECK DATA generates REPAIR statements to a PUNCHDDN data set to invalidate the LOBs/XMLs in the base table space that you can execute later.

## 12.4.2 CHECK INDEX

The CHECK INDEX utility can be run against an auxiliary index to verify that each LOB is represented by an index entry. You can also use the CHECK INDEX utility to check XML indexes and DocID and NodeID indexes, to which we refer further as XML related indexes. Run CHECK INDEX when you suspect there might be inconsistencies between the LOB or XML table space and the indexes. The CHECK INDEX issues warning messages when inconsistencies are found. The CHECK INDEX utility accesses the index and the LOB/XML table space. It does not access the base table space.

CHECK INDEX reports the following errors:

- ▶ Missing LOBs in the LOB table space compared to the auxiliary index
- ▶ Missing entries in the auxiliary index compared to the LOB table space
- ▶ Inconsistencies between the XML table space and the XML related indexes.

The following actions are performed when using CHECK INDEX SHRLEVEL REFERENCE:

- ▶ DB2 drains all SQL writers and sets the LOB table space and auxiliary index in UTRO. It then unloads all the entries from the LOB table space and compares them with the entries in the auxiliary index. Similar actions occur when checking XML related indexes.
- ▶ Invalid entries in the LOB table space can be deleted by using REPAIR LOCATE DELETE and specifying the ROWID and VERSION of the LOB or by doing a point-in-time recovery of the LOB table space to bring it in sync again with the auxiliary index. Invalid entries in the auxiliary index can be removed by doing a REBUILD INDEX of the auxiliary index.
- ▶ Orphan entries in the XML table space can be deleted by using REPAIR LOCATE RID DELETE. Invalid entries in the XML related indexes can be removed by doing a REBUILD INDEX of the XML related index.

The following actions are performed when using CHECK INDEX SHRLEVEL CHANGE:

- ▶ CHECK INDEX runs on the Snapshot data sets, and during its execution, the LOB table space and auxiliary index remain available for SQL writers. It then unloads all the entries from the LOB table space shadow data set and compares them with the entries in the auxiliary index shadow data set. Similar actions occur for the XML objects.

- ▶ Invalid entries in the LOB table space can be deleted by using REPAIR LOCATE DELETE specifying the ROWID and VERSION of the LOB or by doing a point-in-time recovery of the LOB table space to bring it in sync again with the auxiliary index. Invalid entries in the auxiliary index can be removed by doing a REBUILD INDEX.
- ▶ Invalid entries in the XML table space can be deleted by using REPAIR LOCATE RID DELETE. Invalid entries in the XML related indexes can be removed by doing a REBUILD INDEX of the XML related index.

### 12.4.3 CHECK LOB

CHECK LOB can only be run on LOB table spaces, not on XML table spaces. There is no equivalent CHECK XML utility. Run CHECK LOB when you suspect there might be structural defects or when the LOB table space is in the check pending state (CHKP) or auxiliary warning state (AUXW). It reports the following errors:

- ▶ Invalid LOBs: An invalid LOB is the status of a LOB as set by RECOVER when an uncorrected LOB column error is found. CHECK LOB examines the invalid flag in the LOB table space, but it never sets it.
- ▶ Defective LOBs: A defective LOB is a logically inconsistent LOB with a structural defect (most probably as the result of an operational problem that left the LOB table space in an inconsistent state or as the result of a software defect).

CHECK LOB does not access the base table space, so missing LOBs are not detected during CHECK LOB.

The following actions are performed when using CHECK LOB SHRLEVEL REFERENCE:

- ▶ DB2 drains all SQL writers and sets the LOB table space in check pending (CHKP) status in case an error is detected, making it unavailable for all applications. Before APAR PK55972 (DB2 9 only), the LOB table space was already put in CHKP during utility initialization, making it unavailable from the beginning.
- ▶ CHECK LOB issues message DSNU743I whenever it finds a LOB value that is invalid in the LOB table space. The violation is identified by the ROWID and version number of the LOB, a reason code for the error, and the page number where the error was found. Other messages are issued when it detects a defective LOB.
- ▶ If CHECK LOB encounters no more invalid LOBs and no other errors, the LOB table space is reset to a no pending state. If invalid LOB columns remain, CHECK LOB sets the LOB table space to the auxiliary warning (AUXW) status and the table is available for applications.
- ▶ You can use SQL to populate an invalid LOB by updating or deleting the entire row. However, any other attempt to access the column results in a -904 SQL return code (reason code 00C900D0). But the base table and all other non-invalid LOBs are available for applications. If CHECK LOB encountered defective LOBs, the LOB table space is left in CHKP status and is unavailable for applications.

The following actions are performed when using CHECK LOB SHRLEVEL CHANGE:

- ▶ CHECK LOB runs on the Snapshot data set, and during its execution, the LOB table space remains available for applications if it was not yet in the CHKP status. CHECK LOB issues message DSNU743I whenever it finds a LOB value with the invalid flag set in the LOB table space. The violation is identified by the ROWID and version number of the LOB, a reason code for the error, and the page number where the error was found.
- ▶ If you provide a PUNCHDDN data set, the CHECK LOB utility generates REPAIR DELETE statements to delete the bad LOBs afterwards. You can also use SQL to update or delete these LOB columns. CHECK LOB SHRLEVEL CHANGE never sets or resets CHKP or AUXW states on the LOB table space.

For examples and other considerations for checking LOB data, refer to *LOBs with DB2 for z/OS: Stronger and Faster*, SG24-72700.

## 12.5 CHECK of clone tables

Clone tables were introduced in DB2 9 and provide you with the ability to generate a table with the exact same attributes as a base table that already exists. The clone table is created in the same table space as the base table. Once the clone table is created, you can independently work with it (that is you can, for example, load it or insert rows into it).

The clone table is structurally identical to the base table in every way (that is, it has the same number of columns, column names, data types, check constraints, and so on), and it is also created with the same indexes, before triggers, LOB objects, and so on. DB2 creates the clone objects automatically when it creates the clone table. The base table and clone table objects each have separate underlying VSAM data sets identified by their data set instance numbers.

Important restrictions are that the base table must be created in a universal table space and that the base table cannot contain after triggers and may not be involved in referential constraints.

As an example, let us create a table to store images of a certain format (STRING, JPEG, TIF, BMP, and so on) into a BLOB column. The DDL is shown in Example 12-6. We create a unique index on the key of the table and we also create a clone table.

*Example 12-6 DDL to create table with BLOB columns and CLONE*

---

```
CREATE TABLE DB2R6.IMAGES
  (IMAGE_ID    VARCHAR(30) NOT NULL,
   IMPORTER    CHAR(8) FOR SBCS DATA NOT NULL
               WITH DEFAULT USER,
   IMPORT_TIME TIMESTAMP NOT NULL WITH DEFAULT,
   FORMAT      VARCHAR(8),
   DESCRIPTION VARCHAR(255),
   IMAGE       BLOB(1G) WITH DEFAULT NULL) ;
CREATE UNIQUE INDEX DB2R6.IMAGES_INDEX
ON DB2R6.IMAGES (IMAGE_ID) CLUSTER ;
ALTER TABLE DB2R6.IMAGES ADD CLONE IMAGES_CLONE ;
```

---

Because we did not specify a database and table space, a universal table space is created and all underlying auxiliary objects are created automatically. The names generated by DB2 are shown in Table 12-1.

Table 12-1 Automatically created objects

Object	Name
Database	DSN00357
Table space	IMAGES
LOB table space	LCBR036O
Index space	IMAGESRI
Auxiliary index	DB2R6.IIMAGEIMAGECBR0YOZ
Auxiliary index space	IIMAGEIM

The VSAM data sets shown in Example 12-7 are created. The I0001 objects correspond with the base objects, and the I0002 objects with the clone objects.

Example 12-7 VSAM objects created for table and clone table with LOB column

Command - Enter "/" to select action	Message	Volume
-----		
DB9AU.DSNDBC.DSN00357.IIMAGEIM.I0001.A001		*VSAM*
DB9AU.DSNDBC.DSN00357.IIMAGEIM.I0002.A001		*VSAM*
DB9AU.DSNDBC.DSN00357.IMAGES.I0001.A001		*VSAM*
DB9AU.DSNDBC.DSN00357.IMAGES.I0002.A001		*VSAM*
DB9AU.DSNDBC.DSN00357.IMAGESRI.I0001.A001		*VSAM*
DB9AU.DSNDBC.DSN00357.IMAGESRI.I0002.A001		*VSAM*
DB9AU.DSNDBC.DSN00357.LCBR0360.I0001.A001		*VSAM*
DB9AU.DSNDBC.DSN00357.LCBR0360.I0002.A001		*VSAM*

The base table can be populated by SQL statements, as shown in Example 12-8.

Example 12-8 Sample SQL to populate the base table

```

INSERT INTO DB2R6.IMAGES (IMAGE_ID, DESCRIPTION, FORMAT, IMAGE)
INLUES ('IM00001', 'TEST OBJECT', 'STRING', BLOB('IM000001 DATA')) ;
INSERT INTO DB2R6.IMAGES (IMAGE_ID, DESCRIPTION, FORMAT, IMAGE)
VALUES ('IM00002', 'TEST OBJECT', 'STRING', BLOB('IM000002 DATA')) ;
INSERT INTO DB2R6.IMAGES (IMAGE_ID, DESCRIPTION, FORMAT, IMAGE)
VALUES ('IM00003', 'TEST OBJECT', 'STRING', BLOB('IM000003 DATA')) ;
INSERT INTO DB2R6.IMAGES (IMAGE_ID, DESCRIPTION, FORMAT, IMAGE)
VALUES ('IM00004', 'TEST OBJECT', 'STRING', BLOB('IM000004 DATA')) ;
INSERT INTO DB2R6.IMAGES (IMAGE_ID, DESCRIPTION, FORMAT, IMAGE)
VALUES ('IM00005', 'TEST OBJECT', 'STRING', BLOB('IM000005 DATA')) ;

```

A clone table with an SQL statement is shown in Example 12-9.

*Example 12-9 Sample SQL to populate the clone table*

---

```

INSERT INTO DB2R6.IMAGES_CLONE (IMAGE_ID, DESCRIPTION, FORMAT, IMAGE)
VALUES ('IM00006', 'TEST OBJECT', 'STRING', BLOB('IM000006 DATA')) ;
INSERT INTO DB2R6.IMAGES_CLONE (IMAGE_ID, DESCRIPTION, FORMAT, IMAGE)
VALUES ('IM00007', 'TEST OBJECT', 'STRING', BLOB('IM000007 DATA')) ;
INSERT INTO DB2R6.IMAGES_CLONE (IMAGE_ID, DESCRIPTION, FORMAT, IMAGE)
VALUES ('IM00008', 'TEST OBJECT', 'STRING', BLOB('IM000008 DATA')) ;
INSERT INTO DB2R6.IMAGES_CLONE (IMAGE_ID, DESCRIPTION, FORMAT, IMAGE)
VALUES ('IM00009', 'TEST OBJECT', 'STRING', BLOB('IM000009 DATA')) ;

```

---

If you specify CLONE on the CHECK INDEX syntax, DB2 checks only the specified index spaces that are on the clone table (I0002).

If you specify CLONE on the CHECK DATA syntax, DB2 checks only the clone table within the specified table space (I0002). As clone tables cannot have referential constraints, the utility only checks table check constraints and consistencies between the clone table and the corresponding LOB or XML table spaces.

If you specify CLONE on the CHECK LOB syntax, DB2 checks only the LOB table space corresponding with the clone table (I0002).

If we execute the SQL statement EXCHANGE DATA BETWEEN TABLE DB2R6.IMAGES AND DB2R6.IMAGES\_CLONE, the I0002 objects will become the base objects and the I0001 objects the clone objects. This can be verified in the INSTANCE column of table space SYSIBM.SYSTABLESPACE, which will now contain the value '2' and with the -DISPLAY DATABASE(DSN00357) SPACENAM(\*) command, as shown in Example 12-10.

*Example 12-10 Checking the base and clone object with the DISPLAY DATABASE command*

---

```

DSNT360I  -DB9A *****
DSNT361I  -DB9A *   DISPLAY DATABASE SUMMARY
              *   GLOBAL
DSNT360I  -DB9A *****
DSNT362I  -DB9A      DATABASE = DSN00357  STATUS = RW
              DBD LENGTH = 8066
DSNT397I  -DB9A
NAME      TYPE PART  STATUS              PHYERRLO PHYERRHI CATALOG  PIECE
-----
IMAGES    TSB2   0001  RW
IMAGES    TSC1   0001  RW
LCBR0360  LSB2           RW
LCBR0360  LSC1           RW
IIMAGEIM  IXB2           RW
IIMAGEIM  IXC1           RW
IMAGESRI  IXB2    L*    RW
IMAGESRI  IXC1    L*    RW
*****  DISPLAY OF DATABASE DSN00357 ENDED  *****
DSN9022I  -DB9A DSNTDDIS 'DISPLAY DATABASE' NORMAL COMPLETION
***

```

---

Base table objects that have been cloned are noted with the character 'B' in the TYPE column of the DISPLAY output. Cloned objects are noted with the character 'C' in the TYPE column. Immediately following the 'B' or 'C' character is the data set instance number. The data set number is always a '1' or '2'. All the base table objects have the same data set instance number. All of the clone table objects have the same instance number, which is different than the base table objects' instance. Data set instance number associations change during each data exchange command.





## BACKUP and RESTORE SYSTEM utilities

These utilities were introduced with DB2 Version 8 to help customers address issues with critical DB2 systems that require high availability with fast and non-intrusive backup facilities and fast recovery capabilities to minimize down time.

In this chapter, we provide a brief description of DFSMS terms, an overview of the system related functions of BACKUP and RESTORE SYSTEM, and the execution of some scenarios of object-level recovery.

The chapter contains the following sections:

- ▶ DFSMSHsm and DB2 BACKUP and RESTORE SYSTEM
- ▶ BACKUP SYSTEM
- ▶ BACKUP SYSTEM to DASD
- ▶ BACKUP SYSTEM to tape
- ▶ BACKUP SYSTEM additional options
- ▶ New DSNZPARMs
- ▶ Incremental FlashCopy
- ▶ Object-level recovery
- ▶ Scenarios of SYSTEM BACKUP and object-level recovery
- ▶ Considerations for using BACKUP and RESTORE SYSTEM
- ▶ Other available improvements
- ▶ RESTORE SYSTEM
- ▶ Backing up the log Copy Pool prior to system-level recovery

## 13.1 DFSMSHsm and DB2 BACKUP and RESTORE SYSTEM

Storage aware database products allow DBA and system programmers to use fast replication in a safe and transparent manner.

Figure 13-1 shows the integration between database and storage.

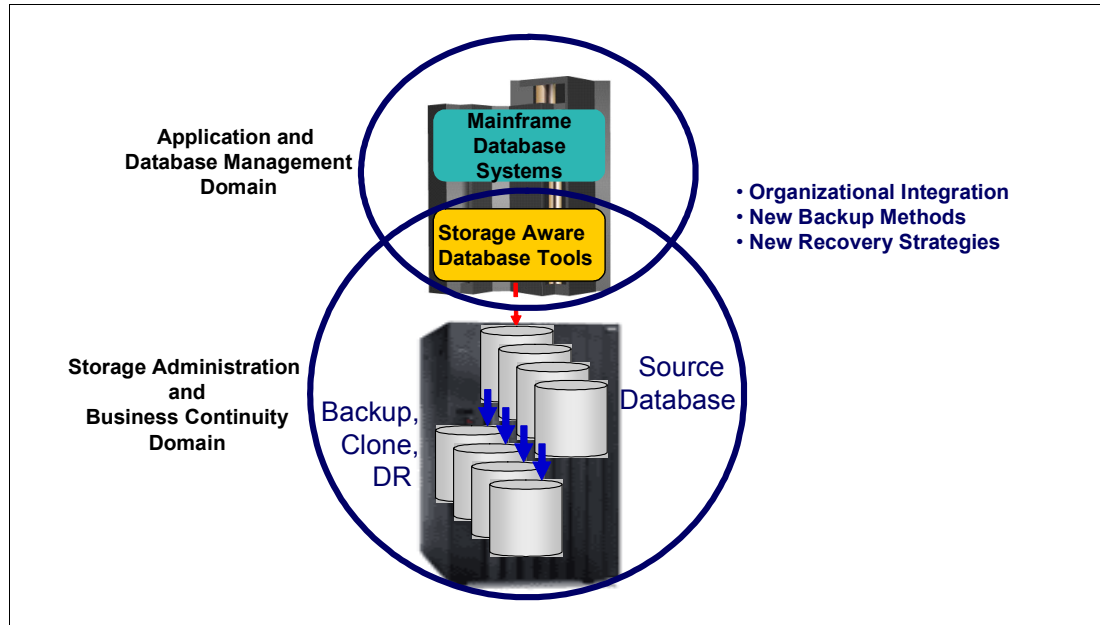


Figure 13-1 Integration between database and storage

The BACKUP SYSTEM is based on DFSMSHsm functions, so some DFSMS terms are used in this chapter. This section provides short descriptions of some of those terms.

### 13.1.1 FlashCopy

A FlashCopy image is an instantaneous copy of a DASD volume taken at a particular point in time. It is possible to keep several versions if resources are available and the data can be maintained on disk or tape and administered by DB2 9.

To take a FlashCopy, you must first establish a relationship between the Copy Pool<sup>1</sup> (source) and the backup Copy Pool (target). Volumes will be logically associated so that a physical copy of each volume can be made. At the point the relationship is established, the BACKUP SYSTEM or RESTORE SYSTEM is considered logically complete.

For example, when doing a BACKUP SYSTEM, the unavailability period for access to the system only lasts until the BACKUP SYSTEM is logically complete, so it is very fast. Once logically complete, a background copy is started so that the target will look like the source did at the time the operation was logically complete. If any applications cause changes to tables so that tracks on the source volume must be updated before they are copied to the target, the source track with the change is first copied to the target before it is updated. Once the copy of each volume is complete, the logical relationship can be terminated.

<sup>1</sup> A set of storage groups

In preparation for FlashCopy usage, the storage team needs to run the DFSMSHsm FRBACKUP PREPARE command to assigns specific target volumes to each source volume for each version of the Copy Pool.

Figure 13-2 shows an example of a FlashCopy relationship.

**Note:** The DFSMSHsm FRBACKUP PREPARE command is recommended when an environment is set up and every time the environment is changed. This command should be run by a storage administrator.

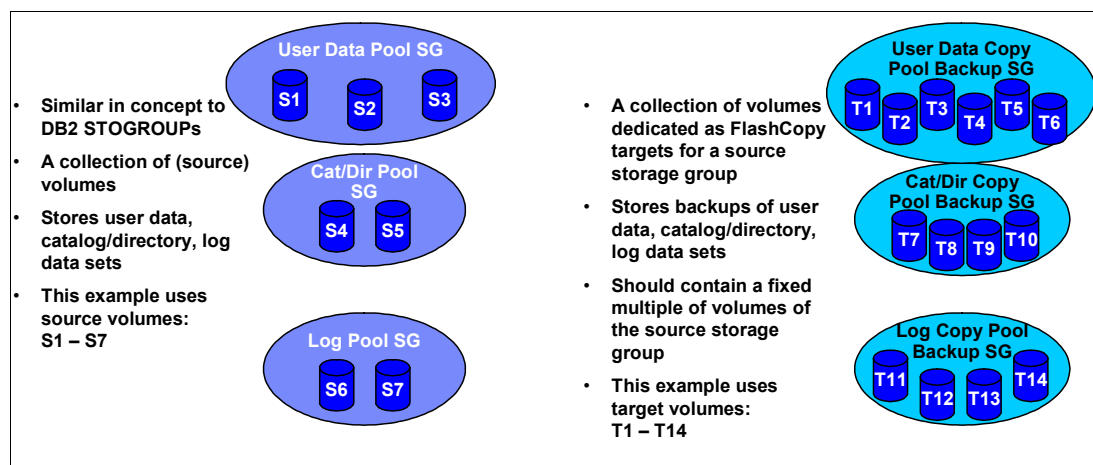


Figure 13-2 FlashCopy relationship

For detailed information about the DFSMSHsm Fast Replication function and FlashCopy, refer to *DFSMSHsm Fast Replication Technical Guide*, SG24-7069.

## 13.1.2 TOKEN

A token is a unique identification for a backup. It is 36 bytes in length. It should be compared to the timestamp functionality.

## 13.1.3 COPYPOOL

A Copy Pool is a set of Storage Management Subsystem (SMS) storage groups that contain data that DFSMSHsm can back up collectively. Once the backup is taken, it is identified by a token.

A Copy Pool can consist of one or more data storage groups, up to a maximum of 256, and it can have up to 85 versions.

## 13.1.4 DUMPCLASS

DUMPCLASS is used when you want to dump to tape. It indicates the DFSMSHsm dumpclass that you want to use for the dump processing. You can specify up to five dumpclasses. If you do not specify a dumpclass, DB2 uses the default dumpclasses that are defined for the Copy Pools.

## 13.2 BACKUP SYSTEM

BACKUP SYSTEM support was introduced in DB2 for z/OS V8 to provide an easier and less disruptive way for fast volume-level backup and recovery. Its implementation was based on the usage of FlashCopy to back up DB2 data and logs, consequently eliminating the need to suspend logs. System-level backups are managed by DB2 and DFSMSHsm to support system level point-in-time (PIT) recovery.

BACKUP SYSTEM works in an online fashion and allows concurrent applications to continue to access the database for read and write operations.

To exploit the BACKUP SYSTEM support, all DB2 data sets and logs must be SMS-managed as well as be part of DB2 DFSMSHsm Copy Pools.

**Important:** If you are planning to use the BACKUP SYSTEM utility, you must have a dedicated DASD for each DB2 subsystem or data sharing group. If you have table spaces from more than one DB2 in the same DASD volume, when you execute a RESTORE SYSTEM for a specific DB2 subsystem, you could have inconsistent data restored.

BACKUP SYSTEM requires that you define two Copy Pools for each DB2 system:

- ▶ A data Copy Pool, containing all application data, and the DB2 catalog and directory and corresponding ICF catalogs
- ▶ A log Copy Pool, containing the DB2 log data sets, BSDS, and corresponding ICF catalogs

The BACKUP SYSTEM gives you the option to either copy both Copy Pools or to copy the data Copy Pool only. If you would like to restore the DB2 system or an object to an arbitrary point in time, copying the data Copy Pool only should be sufficient. On the other hand, if you plan to clone the DB2 at the system level or if you would like restore DB2 to the point when the backup was created, then copying both Copy Pools is required.

The Copy Pools must follow this format:

DSN\$<location-name>\$<cp-type>

Where:

<b>DSN</b>	The unique DB2 product identifier
<b>\$</b>	A fixed character delimiter
<b>&lt;location-name&gt;</b>	The DB2 location name
<b>\$</b>	A fixed character delimiter
<b>&lt;cp-type&gt;</b>	The Copy Pool type.

Use DB for data Copy Pool and LG for log Copy Pool. For example:

```
DATA COPYPOOL (DSN$DB9A$DB)
LOG COPYPOOL (DSN$DB9A$LG)
```

Example 13-1 shows the output of the DB2 display DDF command, where you can see the location name.

Example 13-1 DB2 DDF display

13.06.27	STC17204	DSNL004I	-DB9A DDF START COMPLETE	928
928			<b>LOCATION DB9A</b>	
928			LU USIBMSC.SCPDB9A	
928			GENERICLU -NONE	
928			DOMAIN wtsc63.itso.ibm.com	
928			TCPPORT 12347	
928			SECPORT 12349	
928			RESPORT 12348	
928			IPNAME -NONE	

Figure 13-3 shows the configuration we will be using to demonstrate the examples in this chapter. DB9ACPB is the Copy Pool backup storage group.

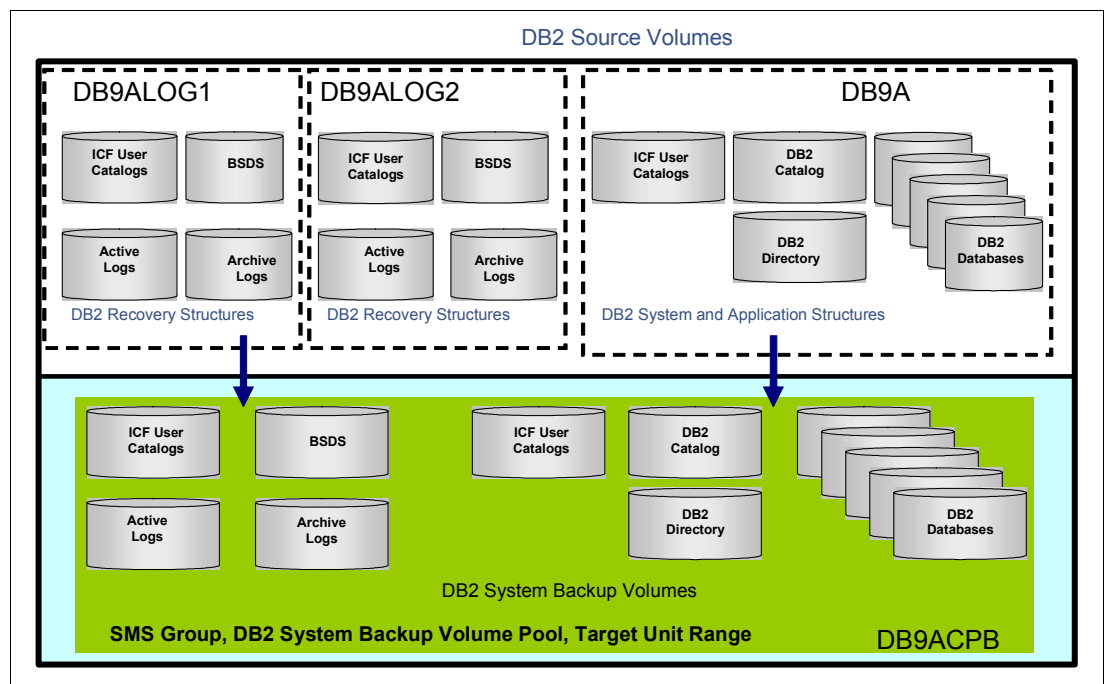


Figure 13-3 Data set layout considerations

As mentioned previously in this chapter, a Copy Pool represents a set of SMS storage groups.

Example 13-2 shows the output from a Copy Pool list, which can be obtained by a storage administrator via ISMF windows, where the Copy Pool DB (DSN\$DB9A\$DB) has one storage group DB9A, and Copy Pool LG has two storage groups, one (DB9ALOG1) for log1 and BSDS1 and the second one (DB9ALOG2) for log2 and BSDS2. This information is under columns 14 and 15.

Example 13-2 List of Copy Pools name

Enter Line Operators below:

LINE OPERATOR	COPY POOL NAME	STORAGE GRP NAME	STORAGE GRP NAME
---(1)---	----- (2) -----	--(14)--	--(15)--
	DSN\$DB9A\$DB	DB9A	-----
	DSN\$DB9A\$LG	DB9ALOG1	DB9ALOG2
-----	-----	-----	BOTTOM OF DATA -----

Example 13-3 shows a list of storage groups from source volumes (DB9A, DB9ALOG1,DB9ALOG2). Column 43 shows a relationship between the source volumes in the DB2 Copy Pools, in which the original data sets used by DB2 are located, and the volumes in the Copy Pool backup storage group (DB9ACPB).

Example 13-3 List of storage groups

STORAGE GROUP LIST

Command ==>

Scroll ==> HALF

Entries 1-4 of 4

View in Use

CDS Name : ACTIVE

Enter Line Operators below:

LINE	STORGRP	SG	VIO	VIO	AUTO	CP BCKP
OPERATOR	NAME	TYPE	MAXSIZE	UNIT	MIGRATE	SG NAME
---(1)---	--(2)---	----- (3) -----	--(4)--	(5)-	--(6)---	--(43)--
	DB9A	POOL	-----	----	NO	DB9ACPB
	DB9ALOG1	POOL	-----	----	NO	DB9ACPB
	DB9ALOG2	POOL	-----	----	NO	DB9ACPB
-----	-----	-----	BOTTOM	OF	DATA	-----

When you use the command LISTVOL, you can view a list of volumes that should be copied, as shown in Example 13-4. Both volumes SBOX5Q and SBOX5R were defined in the DB9A SMS storage group.

*Example 13-4 Volumes for storage group*

---

```

                                VOLUME LIST
Command ==>                                Scroll ==> HALF
                                           Entries 1-2 of 2
Enter Line Operators below:                Data Columns 3-8 of 43

      LINE      VOLUME FREE      %      ALLOC      FRAG      LARGEST      FREE
      OPERATOR   SERIAL SPACE    FREE    SPACE    INDEX    EXTENT    EXTENTS
    --- (1) ---  - (2) --  --- (3) ---  (4) -  --- (5) ---  - (6) -  --- (7) ---  -- (8) --
                        SBOX5Q  4078757K   49   4235744K   67   3570773K    20
                        SBOX5R  3640053K   44   4674448K   90   3033627K    25
    -----  -----  -----  BOTTOM OF DATA  -----  -----  -----

```

---

**Note:** Commands at the line level should not be issued when using views. Views are used when you work at the specific column level.

Example 13-5 shows the DFSMSHsm LIST COPYPOOL job.

*Example 13-5 LIST COPYPOOL job*

---

```

//STEP01 EXEC PGM=IKJEFT01
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSTSIN DD *
      HSEND CMD WAIT LIST COPYPOOL(DSN$DB9C$DB) -
      ODS(DB2R7.TEST2)

```

---

Example 13-6 shows the output of the job in Example 13-5 on page 345. The output reports the following information:

- ▶ The relationship between source and target DASD
- ▶ If one backup was incremental or not
- ▶ Whether the background copy is complete<sup>2</sup>
- ▶ Dumpstate (dumped to tape or not)

*Example 13-6 Relationship between source and target volumes*

---

```

COPYPOOL=DSN$DB9C$DB
ALLOWPPRCP FRB=NO FRR=NO

VERSION  VTOCENQ    DATE          TIME          FASTREPLICATIONSTATE  DUMPSTATE
   042      N      2009/10/08      14:58:28      RECOVERABLE          NONE
TOKEN(C)=C'D9C1DY , DT $ '
TOKEN(H)=X'C4F9C3F1C4E8156B055314CBC4E3145B2232'
TOTAL NUM OF VOLUMES=00004,INCREMENTAL=Y

SGNAME    SOURCE - TARGET  SOURCE - TARGET  SOURCE - TARGET  SOURCE - TARGET
DB9CDATA SBOX5S - SBOX7A  SBOX5T - SBOX7B  SBOX5U - SBOX7C  SBOX5V - SBOX7D

VERSION  VTOCENQ    DATE          TIME          FASTREPLICATIONSTATE  DUMPSTATE
   041      N      2009/10/07      14:21:15      RECOVERABLE          NONE
TOKEN(C)=C'D9C1DW " DT $ '
TOKEN(H)=X'C4F9C3F1C4E6CB3BBD2B1A49C4E3145B2232'
TOTAL NUM OF VOLUMES=00004,INCREMENTAL=N

SGNAME    SOURCE - TARGET  SOURCE - TARGET  SOURCE - TARGET  SOURCE - TARGET
DB9CDATA SBOX5S - SBOX6A  SBOX5T - SBOX6B  SBOX5U - SBOX6C  SBOX5V - SBOX6D

```

---

## 13.3 BACKUP SYSTEM to DASD

The invocation of the BACKUP SYSTEM utility from a JCL job is shown in Example 13-7.

*Example 13-7 BACKUP SYSTEM JCL utility*

---

```

000010 //DB2R7001 JOB CLASS=A,TIME=1440,REGION=0M,
000011 //                MSGCLASS=T,MSGLEVEL=(1,1)
000020 /*JOBPARM S=SC63
000021 //*****
000022 /** STEP BACKUP : RUN BACKUP SYSTEM FOR DB9A                **
000023 //*****
000031 //BACKUP EXEC PGM=DSNUTILB,PARM=(DB9A,BKSYS),REGION=0M
000032 //STEPLIB DD DISP=DHR,DSN=DB9A9.SDSNEXIT
000033 //                DD DISP=SHR,DSN=DB9A9.SDSNLOAD
000034 //SYSOUT DD SYSOUT=*
000035 //SYSPRINT DD SYSOUT=*
000036 //SYSIN DD *
000037 BACKUP SYSTEM FULL

```

---

<sup>2</sup> FASTREPLICATIONSTATE RECOVERABLE means that the background is complete and the volume could be flashed back.



Example 13-8 shows the BACKUP SYSTEM output and the order in which COPYPOOL is executed (the data Copy Pool and then the log Copy Pool). The token that is passed from DB2 to HSM is also displayed. This token can be used to identify backup versions in HSM.

Example 13-8 BACKUP SYSTEM output

```

DSNUGUTC - BACKUP SYSTEM FULL
  DSNUVBBD - BACKUP SYSTEM UTILITY FOR DATA STARTING,
              COPYPOOL = DSN$DB9A$DB
              TOKEN = X'C2C1D740C4DA58F83EE37183D083CC8644E4'.
  DSNUVBBD - BACKUP SYSTEM UTILITY FOR DATA COMPLETED SUCCESSFULLY,
              COPYPOOL = DSN$DB9A$DB
              TOKEN = X'C2C1D740C4DA58F83EE37183D083CC8644E4'
              ELAPSED TIME = 00:00:05.
  DSNUVBBD - BACKUP SYSTEM UTILITY FOR LOGS STARTING,
              COPYPOOL = DSN$DB9A$LG
              TOKEN = X'C2C1D740C4DA58F83EE37183D083CC8644E4'.
  DSNUVBBD - BACKUP SYSTEM UTILITY FOR LOGS COMPLETED SUCCESSFULLY,
              COPYPOOL = DSN$DB9A$LG
              TOKEN = X'C2C1D740C4DA58F83EE37183D083CC8644E4'
              ELAPSED TIME = 00:00:00.
  DSNUVBBD - BACKUP SYSTEM UTILITY COMPLETED, ELAPSED TIME = 00:00:06
  DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0

```

In a data sharing environment, the submitting member records the backup information in its BSDS and also in the SCA. If you want to see backup system entries, run DSNJU004 with the GROUP DD statement pointing to the BSDS data sets. This will merge all the BSDS information from all the members.

Example 13-9 shows the BACKUP SYSTEM information registered on BSDS.

Example 13-9 Output of BSDS

BACKUP SYSTEM UTILITY HISTORY					
SUBSYSTEM ID DB9A					
19:17:42 SEPTEMBER 28, 2009					
START STCK		DATA COMPLETE	DATA/LOG COMPL		
DATA	LOG	RBLP	LRSN	DATE	LTI
-----	-----	-----	-----	-----	-----
C4DAF9F658BEF785	C4DAF9FD1EFE0A83	D084F438F000	D084F438F000	2009/09/28	10:4
	TOKEN = C2C1D740C4DAF9F658BEF785D084F438F000				
C4DA58F83EE37183	C4DA58FD5480C003	D083CC8644E4	D083CC8644E4	2009/09/27	22:4
	TOKEN = C2C1D740C4DA58F83EE37183D083CC8644E4				
C4D9B816CCD4FD05	C4D9B81EAF168E84	D082EADA09F6	D082EADA09F6	2009/09/27	10:4
	TOKEN = C2C1D740C4D9B816CCD4FD05D082EADA09F6				

**Note:** When a TOKEN is deleted from HSM for any reason, DB2 does not know about it.

## 13.4 BACKUP SYSTEM to tape

In DB2 9, the BACKUP SYSTEM utility can copy the data directly to tape. The new options allowing this capability are DUMP and DUMPONLY.

DUMP takes a copy directly to the tape. DUMPONLY takes a copy from the target Copy Pool to tapes.

To have a minimal impact on your (production) system, use BACKUP SYSTEM, then, when the background copy is complete, invoke BACKUP SYSTEM DUMPONLY. Because the background copy is complete, DUMPONLY will minimize the impact to the I/O subsystem for the source/production volumes. Otherwise, both the I/O between the DASD Copy Pools and the I/O between the target Copy Pool and the tapes will be active concurrently.

### 13.4.1 DUMP

This option should be used when you need to create a fast replication copy of the database Copy Pool and log Copy Pool on disk and then initiate a dump to tape of the fast replication copy. The dump to tape begins after DB2 successfully establishes relationships for the fast replication copy. It requires z/OS Version 1.8. You have to have DUMPCCLASS defined to execute this function.

### 13.4.2 DUMPONLY

This option is used when you want to create a dump on tape of an existing fast replication copy (that is currently residing on the disk) of the database Copy Pool and the log Copy Pool. This option requires z/OS Version 1.8.

Keep in mind that directly copying data to tape will have an impact on the speed of your restore. Restoring from tape will not be as fast as restoring from a FlashCopy made to disk. Of course, having your data on tape can help in terms of storage management, disaster recovery, and offsite data storage. Be aware of these tradeoffs before creating system level backups on tape.

The output of DUMP or DUMPONLY is directed to what is called DFSMSHsm *dumpclass*. Each dumpclass will specify the unit type to which the data will be directed.

DB2 allows up to five dumpclasses to be specified. Each specification results in a complete copy of the Copy Pools being backed up and being directed to each dumpclass. When you have multiple dumpclasses, it is possible, for example, to send one copy to remote devices for DR purposes, while keeping another copy for local operational reasons.

The dump to tape is always a full dump even if incremental FlashCopy is used.

The status of the DUMPSTATE column when you issue a DFSMSHsm LIST COPYPOOL command can be one of the following statuses:

- ▶ NONE: The backup for the DB2 data and log Copy Pools only reside on DASD.
- ▶ PARTIAL: The backup has not been completely dumped to tape yet.
- ▶ ALLCOMPLETE: All the volumes have successfully been dumped to tape.

To take a backup on tape, create and associate a dumpclass with your Copy Pool, or you will receive message ARC0635I on MVS log, which states that your dumpclass is not valid, as shown in Example 13-10.

*Example 13-10 DFSMSHsm message on MVS log*

---

```
ARC0635I DUMP OF COPYPOOL DSN$DB9C$DB NOT PERFORMED, 607
ARC0635I (CONT.) REASON=4
```

---

Example 13-11 shows the job output with error messages for the same problem. Notice that the DFSMSHsm messages do not appear on the DB2 job, but on MVS log or HSM STC.

*Example 13-11 Output for DUMP parameter*

---

```
DSNU000I 287 14:32:26.61 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = BKSYS
DSNU1044I 287 14:32:26.65 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
DSNU050I 287 14:32:26.65 DSNUGUTC - BACKUP SYSTEM DUMP
DSNU1600I 287 14:32:27.21 DSNUVBBD - BACKUP SYSTEM UTILITY FOR DATA STARTING,
        COPYPOOL = DSN$DB9C$DB
        TOKEN = X'C4F9C3F1C4EF9ACAA9CAD6CBC4EC221659AB'.
DSNU1631I 287 14:32:27.21 DSNUVBBD - BACKUP SYSTEM UTILITY FAILED BECAUSE THE CALL TO
DFSMSHSM FAILED WITH RC =
X'00000018'
REASON = X'00000008'.
        SEE THE HSM ACTIVITY LOG FOR HSM MESSAGES INDICATING THE CAUSE OF
THE ERROR.
DSNU012I 287 14:32:27.82 DSNUGBAC - UTILITY EXECUTION TERMINATED, HIGHEST RETURN CODE=8
```

---

In Figure 13-14 on page 366, scenario number 9 shows one dump taken to tape.

Restore from tape dumps will use parallelism that is limited by the number of distinct tape volumes that the dump resides on, capped by tape units.

**Important:** Given the BACKUP SYSTEM utility's design, it is better to take the backup to tape with the DUMPONLY option rather than the DUMP option, because if any problem with HSM occurs, DB2 will not register the token in the BSDS and you will lose the correct time to take the backup. This occurs because DB2 works in a stand-alone fashion from HSM.

## 13.5 BACKUP SYSTEM additional options

This section discusses additional BACKUP SYSTEM options.

### 13.5.1 DATA ONLY

This option indicates that you want to copy only the database Copy Pool.

Remember that the logical part of the copy is created in a matter of seconds. However, the physical creation of the copy in the backup storage groups is asynchronously performed via a background I/O process. Depending on the size of the database, this phase of FlashCopy processing may take minutes or hours.

If a track on the source volume changes before the background copy is complete, then that track will be copied to the target volume before the new contents of the track is written to the source volume.

## 13.5.2 FORCE

The FORCE option can only be used when creating a backup on tape. When creating a backup on DASD, the utility will ignore this option and take another BACKUP SYSTEM.

This option should be used when you decide to overwrite the oldest DFSMSHsm version copy of Copy Pool. You can overwrite these Copy Pools if the dump to tape has been initiated but is only partially completed.

For example, if the BACKUP SYSTEM with the oldest DASD version is still in the process of being dumped to tape, but you need to take a new BACKUP SYSTEM on DASD and you are willing to sacrifice the oldest DASD version even though it has not been saved to tape yet, then you should use the FORCE option.

Example 13-12 shows that there is no extra information showing that this backup overwrote the oldest one.

### *Example 13-12 BACKUP FORCE*

---

```
18:49:54.42 DSNUGUTC - BACKUP SYSTEM FORCE
18:49:54.92 DSNUVBBD - BACKUP SYSTEM UTILITY FOR DATA STARTING,
                  COPYPOOL = DSN$DB9C$DB
                  TOKEN = X'C4F9C3F1C4EE92794AF0F086C4EC221659AB'.
18:49:55.63 DSNUVBBD - BACKUP SYSTEM UTILITY FOR DATA COMPLETED SUCCESSFULLY,
                  COPYPOOL = DSN$DB9C$DB
                  TOKEN = X'C4F9C3F1C4EE92794AF0F086C4EC221659AB'
                  ELAPSED TIME = 00:00:00.
18:49:55.63 DSNUVBBD - BACKUP SYSTEM UTILITY FOR LOGS STARTING,
                  COPYPOOL = DSN$DB9C$LG
                  TOKEN = X'C4F9C3F1C4EE92794AF0F086C4EC221659AB'.
18:50:04.97 DSNUVBBD - BACKUP SYSTEM UTILITY FOR LOGS COMPLETED SUCCESSFULLY,
                  COPYPOOL = DSN$DB9C$LG
                  TOKEN = X'C4F9C3F1C4EE92794AF0F086C4EC221659AB'
                  ELAPSED TIME = 00:00:09.
18:50:04.97 DSNUVBBD - BACKUP SYSTEM UTILITY COMPLETED, ELAPSED TIME = 00:00:10.
18:50:04.98 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

---

### 13.5.3 TOKEN

This option can only be used when dumping to tape and should be used when you want to specify the correct BACKUP SYSTEM that you want to dump to tape; otherwise, the most recent copy of the Copy Pools (DB and LOG) will be dumped. The utility DSNJU004 reports the token that uniquely identifies the system-level backup, as shown in Example 13-9 on page 347. Example 13-13 shows how to use this parameter. If you do not specify the TOKEN option, the most recent fast replication copy of the Copy Pools is dumped to tape.

#### *Example 13-13 TOKEN parameter*

---

```
8:36.09 DSNUGUTC - BACKUP SYSTEM DUMPONLY TOKEN=X'C4F9C3F1C4EF83491F4AD44BC4EC221659AB'
8:36.09 DSNUVBBD - BACKUP SYSTEM UTILITY FOR DATA STARTING,
      COPYPOOL = DSN$DB9C$DB
      TOKEN = X'C4F9C3F1C4EF83491F4AD44BC4EC221659AB'.
8:36.09 DSNUVBBD - BACKUP SYSTEM UTILITY FOR DATA COMPLETED SUCCESSFULLY,
      COPYPOOL = DSN$DB9C$DB
      TOKEN = X'C4F9C3F1C4EF83491F4AD44BC4EC221659AB'
      ELAPSED TIME = 00:00:00.
8:36.09 DSNUVBBD - BACKUP SYSTEM UTILITY FOR LOGS STARTING,
      COPYPOOL = DSN$DB9C$LG
      TOKEN = X'C4F9C3F1C4EF83491F4AD44BC4EC221659AB'.
8:36.10 DSNUVBBD - BACKUP SYSTEM UTILITY FOR LOGS COMPLETED SUCCESSFULLY,
      COPYPOOL = DSN$DB9C$LG
      TOKEN = X'C4F9C3F1C4EF83491F4AD44BC4EC221659AB'
      ELAPSED TIME = 00:00:00.
8:36.10 DSNUVBBD - BACKUP SYSTEM UTILITY COMPLETED, ELAPSED TIME = 00:00:00.
8:36.10 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

---

## 13.6 New DSNZPARMs

There are four new parameters in DSNZPARM to control BACKUP and RESTORE SYSTEM:

- ▶ Restore\_recover\_fromdump
- ▶ Restore\_tapeunits
- ▶ System\_level\_backup
- ▶ Utils\_dump\_class\_name

For details, see Table 14-8 on page 402.

## 13.7 Incremental FlashCopy

The incremental FlashCopy hardware feature will reduce the amount of data that has to be processed in the background because only changed tracks are copied.

All tracks on the source volume are considered to be changed when the relationship is established, so all tracks are physically copied on the source volume. Subsequent incremental FlashCopy will copy only the tracks that have changed on the source volume since the last copy was taken.

This feature requires z/OS Version 1 Release 8 DFSMSHsm, with PTF UA28767 in z/OS and UK28089 in DB2 Version 9.

In order for incremental FlashCopy to be used, the relationship must first be established by running the ESTABLISH FCINCREMENTAL command. The relationship can be terminated by running the ENDFCINCREMENTAL command on BACKUP SYSTEM utility.

Example 13-14 shows how to create the relationship.

*Example 13-14 Example of Incremental BACKUP SYSTEM*

---

```
//BACKUP EXEC PGM=DSNUTILB,PARM=(DB9A,BKSYS),REGION=0M
//STEPLIB DD DISP=DHR,DSN=DB9A9.SDSNEXIT
// DD DISP=SHR,DSN=DB9A9.SDSNLOAD
//SYSOUT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
BACKUP SYSTEM ESTABLISH FCINCREMENTAL
```

---

Example 13-15 shows the output for this execution.

*Example 13-15 Output for Incremental BACKUP SYSTEM*

---

```
DSNUGUTC - BACKUP SYSTEM FULL ESTABLISH FCINCREMENTAL
DSNUVBBD - BACKUP SYSTEM UTILITY FOR DATA STARTING,
            COPYPOOL = DSN$DB9A$DB
            TOKEN = X'E2D5F6F1C29ED4388D5B2A46C29ED30F6510'.
DSNUVBBD - THE SYSTEM LEVEL BACKUP TAKEN IS AN INCREMENTAL FLASHCOPY OF
            THE DATABASE
            COPYPOOL = DSN$DB9A$DB
            TOKEN = X'E2D5F6F1C29ED4388D5B2A46C29ED30F6510'
            ELAPSED TIME = 00:00:05.
DSNUVBBD - BACKUP SYSTEM UTILITY FOR LOGS STARTING,
            COPYPOOL = DSN$DB9A$LG
            TOKEN = X'E2D5F6F1C29ED4388D5B2A46C29ED30F6510'.
DSNUVBBD - BACKUP SYSTEM UTILITY FOR LOGS COMPLETED SUCCESSFULLY,
            COPYPOOL = DSN$DB9A$LG
            TOKEN = X'E2D5F6F1C29ED4388D5B2A46C29ED30F6510'
            ELAPSED TIME = 00:00:00.
DSNUVBBD - BACKUP SYSTEM UTILITY COMPLETED, ELAPSED TIME = 00:02:06
```

---

**Attention:** The incremental FlashCopy works differently for DB2 incremental image copy: The tracks override the previous version, and then the resulting backup is always a “full” backup.

### 13.7.1 Incremental FlashCopy with versions

The incremental FlashCopy behavior is different depending on whether you work with a single DASD version or multiple DASD versions. The following examples will explain this behavior. You should define how many versions your Copy Pool will have when you created the Copy Pool.

Example 13-16 shows the output from a Copy Pool specification that verifies how many versions your Copy Pool has.

Example 13-16 Output from HSM Copy Pool

Enter Line Operators below:

LINE		#	BACKUP	AUTO	DUMP	SYS
OPERATOR	COPY POOL NAME	VERSIONS	DUMP	OR	SYS	GRP
---(1)---	-----(2)-----	--(3)---	(4)-	---	(5)---	
	DSN\$DB9B\$DB	1	NO			
	DSN\$DB9C\$DB	2	NO		-----	
	DSN\$DB9C\$LG	2	NO		-----	
-----	-----	-----	BOTTOM	OF	DATA	-----

Figure 13-4 and Figure 13-5 on page 354 give a brief explanation about how BACKUP SYSTEM works with HSM. These figures show the different times it takes to make the copies.

- ▶ T0 is a full backup.
- ▶ T1 is the first incremental, and therefore all pages are copied. This takes as long as the full backup.
- ▶ T2 is an incremental, and this completes much faster.
- ▶ Similarly, T3 and T4 are incremental backups with a shorter duration than a full backup.
- ▶ T5 is the backup that withdraws the relationship. It is still an incremental backup.
- ▶ T6 and T7 are full backups.

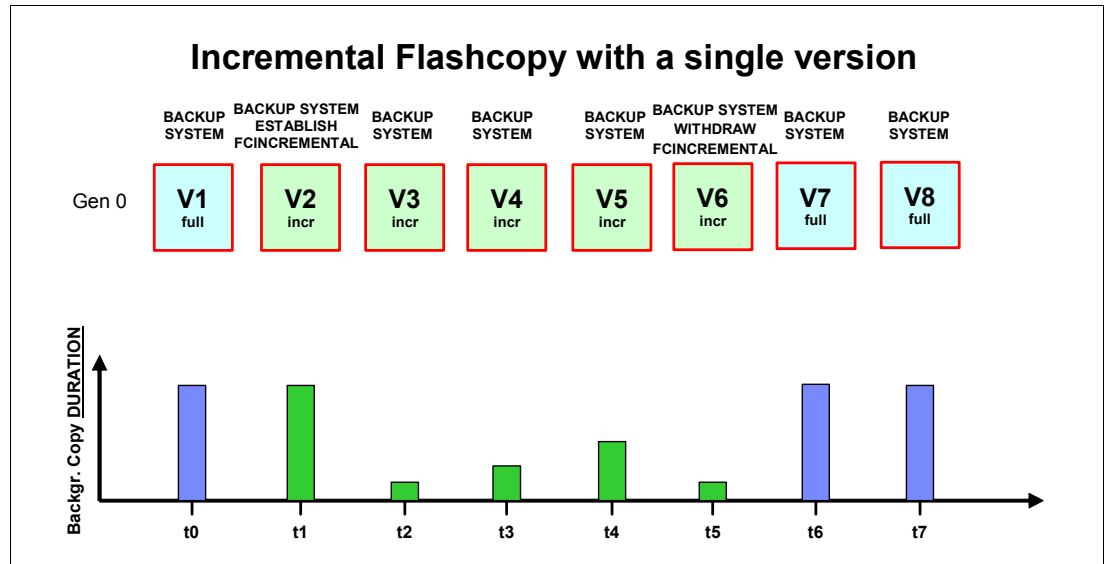


Figure 13-4 Incremental FlashCopy with a single version

Figure 13-5 shows what happens when the Copy Pool is set up for two versions. For each source volume, you have two sets of target volumes to which you can copy. However, you cannot have incremental FlashCopy for both versions of the target volumes, as we are restricted to a single incremental relationship per volume in one Copy Pool.

- At T0, the relationship was established.  
A incremental copy was taken, and this is the first incremental after the relationship has been established. In reality, a full copy is taken to target 1 (V1), and target 2 is empty because this is the first time a copy was made. As you can see, you do not necessarily need to start with BACKUP SYSTEM FULL.
- At T1, the relationship persists and BACKUP SYSTEM is taken.  
Without versioning, this backup results in copying only the changed tracks and is flagged as incremental backup. With versioning, and considering that V1 is already flagged as incremental, target 2 (V2) is a full copy.
- This behavior will continue until the incremental relationship is withdrawn, which occurred at T4, where V4 is kept as full and V5 is taken as an incremental.
- At T5, the relationship terminates and a BACKUP SYSTEM was taken. V5 is kept incremental and V6 is taken as full.

At this point, all versions will be full until a new incremental relationship is created.

Remember that every time that one backup is taken, HSM switches the set of volumes. If you try to force run T3 with BACKUP SYSTEM ESTABLISH FCINCREMENTAL, you will receive the following error:

ARC1806E RC=0062 Version not eligible for FlashCopy incremental

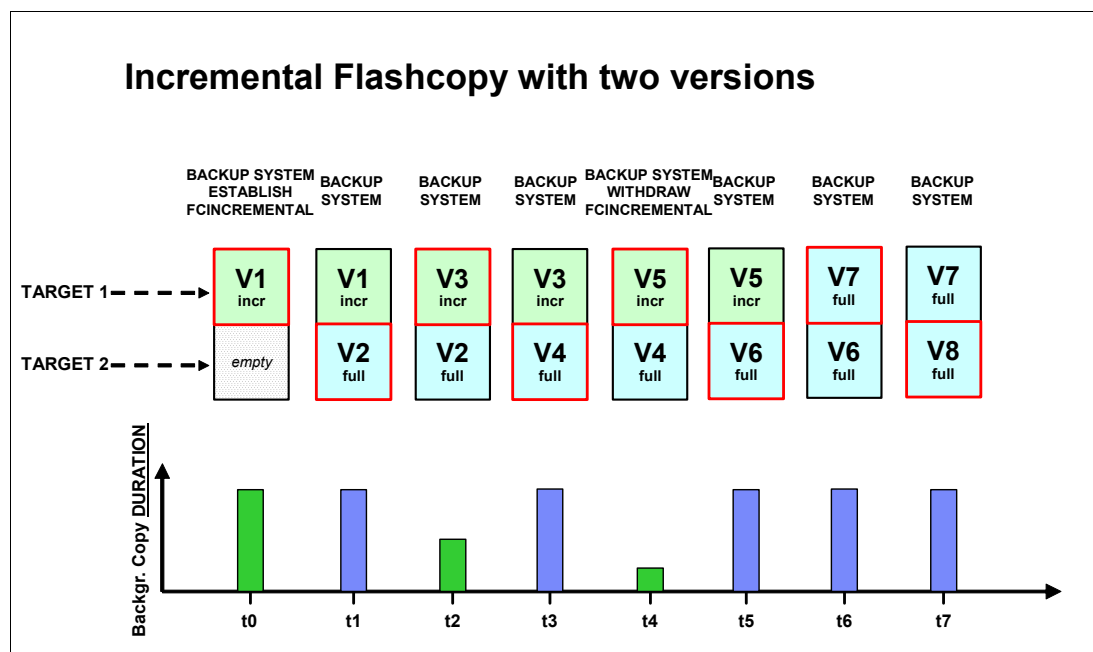


Figure 13-5 Incremental FlashCopy with two versions



## 13.8 Object-level recovery

DB2 9 has an enhancement that allows a subset of the data to be recovered from the BACKUP SYSTEM.

Do not forget that if you would like to use this option you have to set a new DSNZPARM parameter called SYSTEM\_LEVEL\_BACKUPS to YES in window DSNTIP6.

The RECOVER utility is used to execute this recovery. The user will specify the table spaces or indexes in a list, individually, or via a LISTDEF. The most recent recovery base will be determined. In addition to the image copies recorded in the SYSCOPY (or the log for some catalog or directory spaces), the system-level backups are also examined. DFSMSHsm will be used if the last copy before the recovery base is a system-level volume backup.

**Note:** You need to ALTER your indexes to set the COPY YES attribute to enable RECOVER from system-level-backups.

If RECOVER cannot successfully restore from BACKUP SYSTEM, then the utility will terminate, assuming that OPTION EVENT(ITEMERROR,SKIP) was not specified. DB2 will not try to obtain the earlier system-level backup unless the RESTOREBEFORE option on RECOVER is specified.

**Consideration:** If a data set has moved to a different volume or the object was dropped between the time of the BACKUP SYSTEM utility and the time of the RECOVER utility, then object-level recovery is only possible if the image copy is available.

We recommend that image copies are not dispensed with entirely.

**Attention:** If a system-level backup is taken, then an online REORG is run on table space A (ts\_A), and an inline copy must be taken during the REORG to establish a new recovery base for ts\_A. The inline copy allows ts\_A to be fully recovered or to be recovered to a point in time after the REORG. In this case, recovery of ts\_A to a point in time between the BACKUP SYSTEM and the REORG is not allowed. The RECOVER utility will issue msgDSNU1528I - object CANNOT USE THE SYSTEM-LEVEL BACKUP WITH TOKEN X'...' AS A RECOVERY BASE.

This restriction exists because DFSMSHsm is invoked for object-level recovery from a BACKUP SYSTEM and it is a DFSMSHsm requirement that the data sets to be restored must currently reside on the same volumes as when the backup was taken. DFSMSHsm uses the information in the ICF catalog to determine where the data sets resides in the backup, and cannot handle data sets that have “moved” since the backup. (Technically, the online REORG did not move the data sets; it switched to the shadow with the new IPREFIX, and deleted the original data sets with the old IPREFIX.)

This restriction has been somewhat lifted in z/OS V1.11 in that DFSMSHsm can handle the restoration of data sets that have “moved” since the backup was taken, but DFSMSHsm still requires that there be enough space on the original DB2 production volumes (where the object resided at the time of the backup) to restore the data sets.

## 13.9 Scenarios of SYSTEM BACKUP and object-level recovery

In this section, we show some scenarios using BACKUP SYSTEM FULL and INCREMENTAL FlashCopy and object-level recovery. All scenarios are in sequential order of execution.

### Scenario 1

Figure 13-6 shows scenario number 1 where one simple full image copy was taken (T1), then some inserts and updates on table (T2) took place. After that, one BACKUP SYSTEM (T3) was taken, followed by other inserts and updates (T4), one INCREMENTAL FlashCopy (T5), and finally one recovery object level (T6).

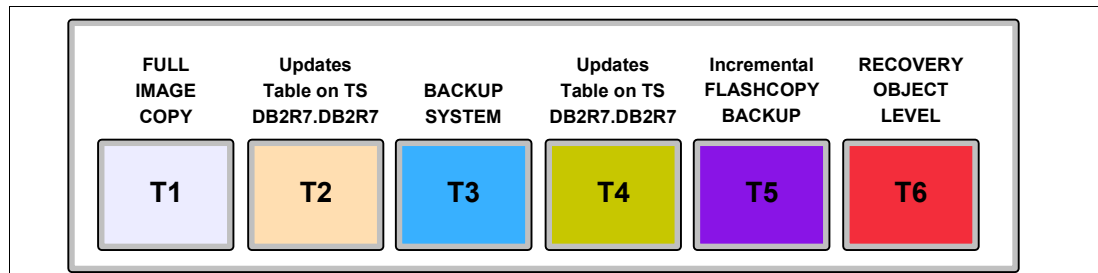


Figure 13-6 Scenario 1

Example 13-17 shows the output for one FULL Image copy (T1) which appears together with message DSNU428I.

#### Example 13-17 Output for backups

---

```

DB2R7.D9C1.IC1.DB2R7.DB2R7          CATALOGED
13:39:35.20 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = DB2R7.DB2R7001
13:39:35.24 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
13:39:35.30 DSNUGUTC - COPY TABLESPACE DB2R7.DB2R7 DSNUM ALL FULL YES
13:39:35.44 DSNUBBID - COPY PROCESSED FOR TABLESPACE DB2R7.DB2R7
      NUMBER OF PAGES=3
      AVERAGE PERCENT FREE SPACE PER PAGE = 25.33
      PERCENT OF CHANGED PAGES = 33.33
      ELAPSED TIME=00:00:00
279 13:39:35.63 DSNUBAFI - DB2 IMAGE COPY SUCCESSFUL FOR TABLESPACE DB2R7.DB2R7
13:39:35.64 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0

```

---

Example 13-18 shows the output of BACKUP SYSTEM (T3).

#### Example 13-18 Output of SYSTEM BACKUP

---

```

13:40:36.61 DSNUGUTC - BACKUP SYSTEM
13:40:37.17 DSNUVBBD - BACKUP SYSTEM UTILITY FOR DATA STARTING,
      COPYPOOL = DSN$DB9C$DB
      TOKEN = X'C4F9C3F1C4E58049024A8A0BC4E3145B2232'.
13:40:37.41 DSNUVBBD - BACKUP SYSTEM UTILITY FOR DATA COMPLETED SUCCESSFULLY,
      COPYPOOL = DSN$DB9C$DB
      TOKEN = X'C4F9C3F1C4E58049024A8A0BC4E3145B2232'
      ELAPSED TIME = 00:00:00.

```

---

Example 13-19 shows the output for incremental system FlashCopy (T5).

*Example 13-19 Incremental system backup output*

---

```
13:41:37.70 DSNUGUTC - BACKUP SYSTEM ESTABLISH FCINCREMENTAL
13:41:38.26 DSNUVBBD - BACKUP SYSTEM UTILITY FOR DATA STARTING,
                  COPYPOOL = DSN$DB9C$DB
                  TOKEN = X'C4F9C3F1C4E5808344FF8846C4E3145B2232'.
13:41:38.97 DSNUVBBD - THE SYSTEM LEVEL BACKUP TAKEN IS AN INCREMENTAL FLASHCOPY

13:41:38.97 DSNUVBBD - BACKUP SYSTEM UTILITY FOR DATA COMPLETED SUCCESSFULLY,
                  COPYPOOL = DSN$DB9C$DB
                  TOKEN = X'C4F9C3F1C4E5808344FF8846C4E3145B2232'
                  ELAPSED TIME = 00:00:00.
```

---

Example 13-20 shows the RECOVER output from the DB2R7.DB2R7 table space that was made using the token from Incremental FlashCopy. When, for the first time, you ESTABLISH FCINCREMENTAL, DFSMSHsm copies all tracks from the source DASD, then RECOVER can use that incremental copy instead of BACKUP SYSTEM FULL.

*Example 13-20 Output of RECOVER*

---

```
DSNU050I   279 13:43:19.69 DSNUGUTC - RECOVER TABLESPACE DB2R7.DB2R7 DSNUM ALL
DSNU1520I   279 13:43:19.77 DSNUCBMT - THE RECOVERY BASE FOR TABLESPACE DB2R7.DB2R7 IS THE SYSTEM LEVEL
BACKUP WITH DATE = 20091006, TIME 134148, AND TOKEN X'C4F9C3F1C4E5808344FF8846C4E3145B2232'
DSNU1527I   279 13:43:20.15 DSNUCBMT - TABLESPACE DB2R7.DB2R7 WAS SUCCESSFULL RESTORED FROM A FLAHCOPY
TIME=00:00:00
DSNU578I   -D9C1 279 13:43:20.18 DSNUCALA - SYSLGRNX INFORMATION FOR MEMBER D9C1
DSNU513I   -D9C1 279 13:43:20.18 DSNUCALA - RECOVER UTILITY LOG APPLY RANGE IS RBA 000134DA9882 LRSN
                  RBA 000134DAC464 LRSN C4E56F761AB7
DSNU513I   -D9C1 279 13:43:20.18 DSNUCALA - RECOVER UTILITY LOG APPLY RANGE IS RBA 000134DB1F08 LRSN
                  RBA 000134DBF000 LRSN C4E570E7C54B
DSNU513I   -D9C1 279 13:43:20.18 DSNUCALA - RECOVER UTILITY LOG APPLY RANGE IS RBA 000134DBF658
                  RBA 000134DD7513 LRSN C4E57179EFAC
DSNU513I   -D9C1 279 13:43:20.18 DSNUCALA - RECOVER UTILITY LOG APPLY RANGE IS RBA 000134DE3344
                  RBA 000134E64A89 LRSN C4E574424E57
DSNU513I   -D9C1 279 13:43:20.18 DSNUCALA - RECOVER UTILITY LOG APPLY RANGE IS RBA 000134E656D6
                  RBA 000134EE790A LRSN C4E575D2EBFF
DSNU513I   -D9C1 279 13:43:20.18 DSNUCALA - RECOVER UTILITY LOG APPLY RANGE IS RBA 000134EE8072
```

---

Example 13-21 shows an excerpt of the output from a RECOVER made for one table space that was not changed in scenario 1. In this case, DB2 used the same token from the incremental system copy, because all tracks on the source volume are considered to be changed when the relationship is established, so all tracks are copied.

Example 13-21 Output of RECOVER

```
RECOVER TABLESPACE GLWDSHR.GLWSPRJ DSNUM ALL
  THE RECOVERY BASE FOR TABLESPACE GLWDSHR.GLWSPRJ  IS THE SYSTEM LEVEL BACKUP
TOKEN X'C4F9C3F1C4E5808344FF8846C4E3145B2232'
  TABLESPACE GLWDSHR.GLWSPRJ  WAS SUCCESSFULLY RESTORED FROM A FLASHCOPY .....

LA - FAST LOG APPLY WAS NOT USED FOR RECOVERY
  LOG APPLY PHASE COMPLETE, ELAPSED TIME = 00:00:00
  RECOVERY COMPLETE, ELAPSED TIME=00:00:00
  REBUILD INDEX(ALL) TABLESPACE GLWDSHR.GLWSPRJ
  INDEXES WILL BE BUILT IN PARALLEL, NUMBER OF TASKS = 7
UL - UNLOAD PHASE STATISTICS - NUMBER OF RECORDS PROCESSED=22908
  UNLOAD PHASE COMPLETE - ELAPSED TIME=00:00:00
```

### Scenario 2

Figure 13-7 shows the scenario 2, where one ALTER table, including one new column on DB2R7.DB2R7 table space (T7), is made, and there is one recovery object level (T8).

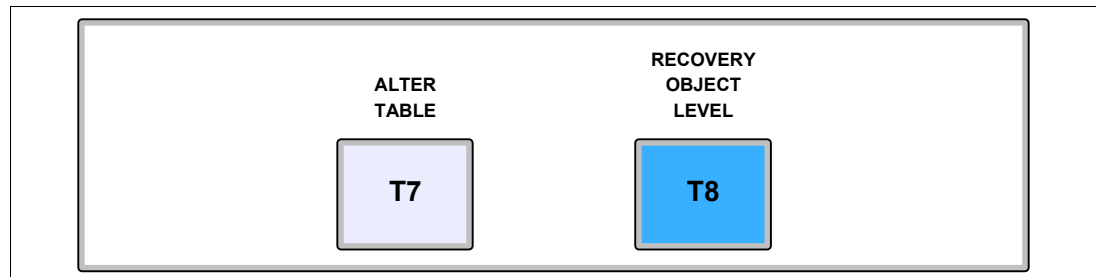


Figure 13-7 Scenario 2

Example 13-22 shows that the DSNJU004 utility was run to open the BSDS display in order to discover the LSRN of BACKUP SYSTEM FULL.

Example 13-22 Output from BSDS

BACKUP SYSTEM UTILITY HISTORY						
SUBSYSTEM ID D9C1						
20:46:42 OCTOBER 06, 2009						
DATA	START STCK	LOG	RBLP	DATA COMPLETE LRSN	DATA/LOG DATE	COMPL LTI
C4E5808344FF8846	C4E58083F2173A4E	C4E3145B2232	C4E5808CD718	2009/10/06	13:4	
TOKEN = C4F9C3F1C4E5808344FF8846C4E3145B2232 TYPE=I						
C4E58049024A8A0B	C4E580493D285B4C	C4E3145B2232	<b>C4E580521D7C</b>	2009/10/06	13:4	
TOKEN = C4F9C3F1C4E58049024A8A0BC4E3145B2232						

Example 13-23 shows the output for Recovery Logpoint to LRSN X'C4E580521D7C', which is a point in time prior to the ALTER.

Remember that this recovery will not modify DDL from the DB2 table. DB2 could manage this using data versioning. In this case, after the ALTER command runs, the values in SYSIBM.SYSTABLESPACE are column OLD\_VERSION with 0 and CURRENT VERSION with 1.

*Example 13-23 Output for Recovery Logpoint*

---

```

DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = DB2R7.DB2R7001
DSNUGTIS - PROCESSING SYSIN AS EBCDIC
DSNUGUTC - RECOVER TABLESPACE DB2R7.DB2R7 DSNUM ALL TOLOGPOINT X'C4E580521D7C'
DSNUCBMT - THE RECOVERY BASE FOR TABLESPACE DB2R7.DB2R7 IS THE SYSTEM LEVEL BA
, AND TOKEN X'C4F9C3F1C4E58049024A8A0BC4E3145B2232'
DSNUCBMT - TABLESPACE DB2R7.DB2R7 WAS SUCCESSFULLY RESTORED FROM A FLASHCOPY,

.68 DSNUCARS - INDEX DB2R7.MARCELOX IS IN REBUILD PENDING
.68 DSNUCARS - ALL INDEXES OF DB2R7.DB2R7 ARE IN REBUILD PENDING
.92 DSNUCALA - SYSLGRNX INFORMATION FOR MEMBER D9C1
.92 DSNUCALA - RECOVER UTILITY LOG APPLY RANGE IS RBA 000134DA9882 LRSN C4E56F76
RBA 000134DAC464 LRSN C4E56F761AB7
.92 DSNUCALA - RECOVER UTILITY LOG APPLY RANGE IS RBA 000134DB1F08 LRSN C4E56FA7
RBA 000134DBF000 LRSN C4E570E7C54B
.92 DSNUCALA - RECOVER UTILITY LOG APPLY RANGE IS RBA 000134DBF658 LRSN C4E570F0
RBA 000134DD7513 LRSN C4E57179EFAC
.92 DSNUCALA - RECOVER UTILITY LOG APPLY RANGE IS RBA 000134DE3344 LRSN C4E57213
RBA 000134E64A89 LRSN C4E574424E57

```

---

**Scenario 3**

Figure 13-8 shows scenario 3, where one BACKUP SYSTEM is taken (T9), the table space is moved to another DASD (T10), and there is one recovery object level (T11).

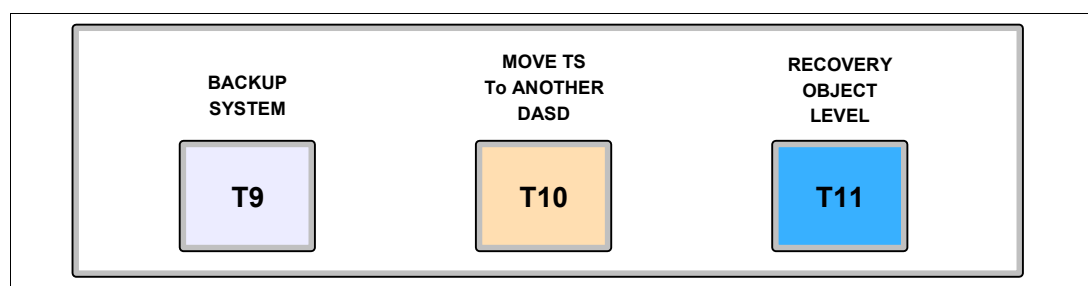


Figure 13-8 Scenario 3

Example 13-24 shows the error message returned to the DB2 job when you are trying to recover one object that was moved to another DASD volume.

*Example 13-24 DB2 output for recovery*

---

```

18:41:47.81 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
18:41:47.82 DSNUGUTC - RECOVER TABLESPACE DB2R7.DB2R7 DSNUM ALL
18:41:47.85 DSNUCBMT - THE RECOVERY BASE FOR TABLESPACE DB2R7.DB2R7   IS THE SYS
    TIME 142120, AND TOKEN X'C4F9C3F1C4E6CB3BBD2B1A49C4E3145B2232'
18:41:48.12 DSNUCBMT - THE DFSMSHSM CALL TO RESTORE TABLESPACE DB2R7.DB2R7
    FAILED WITH RC = X'00000008' AND REASON CODE = X'00000000'
    SEE THE JOB LOG FOR DFSMSHSM MESSAGES INDICATING THE CAUSE OF THE ERROR
280 18:41:48.13 DSNUGSRX - TABLESPACE DB2R7.DB2R7 IS IN RECOVER PENDING
18:41:48.13 DSNUCBDR - RECOVERY COMPLETE, ELAPSED TIME=00:00:00
18:41:48.14 DSNUGBAC - UTILITY EXECUTION TERMINATED, HIGHEST RETURN CODE=8

```

---

Example 13-25 shows the message ARC0624I with rc=8, meaning that no FlashCopy was taken for this table space since it was moved to this DASD volume. If this movement of table space is done outside DB2, you receive a DFSMSHsm message; otherwise, you receive message DSNU556I (when this movement is done, for example, by REORG). Remember that you have to look in the MVS log or DFSMSHsm STC for ARC messages.

*Example 13-25 DFSMSHsm messages when DASD was changed*

---

```

ARC0624I PHYSICAL DATA SET COPY OF VOLUME 931
ARC0624I (CONT.) DB9CD.DSNDBC.DB2R7.DB2R7.I0001.A001 TERMINATED PRIOR
ARC0624I (CONT.) TO COMPLETION, DFSMSDSS FAILING RC = 8
ARC1860I THE FOLLOWING 0001 DATA SET(S) FAILED DURING 932
ARC1860I (CONT.) FAST REPLICATION DATA SET RECOVERY:
ARC1860I (CONT.) DB9CD.DSNDBC.DB2R7.DB2R7.I0001.A001, COPYPOOL=DSN$DB9
C$DB, DEVTYPE=DASD, VOLUME=SBOX5U, ARC1166, RC=0008
ARC1802I FAST REPLICATION DATA SET RECOVERY HAS 934
ARC1802I (CONT.) COMPLETED FOR DATA SET DB9CD.DSNDBC.DB2R7.DB2R7.**
ARC1802I (CONT.) AT 18:41:48 ON 2009/10/07, FUNCTION RC=0008, MAXIMUM
ARC1802I (CONT.) DATA SET RC=0066

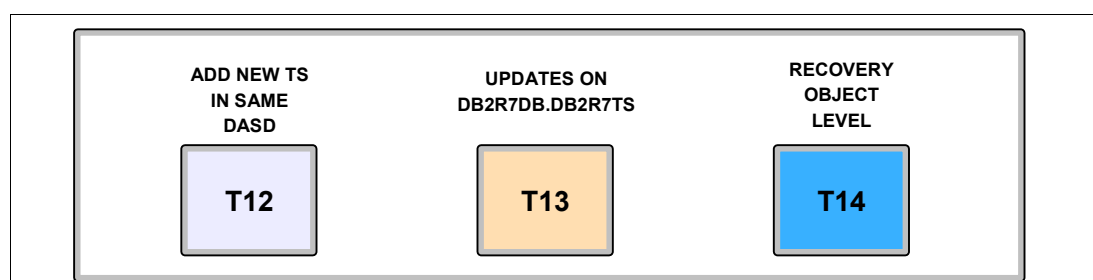
```

---

**Important:** Events that move data sets invalidate the use of existing system-level backups for point-in-time recovery.

## Scenario 4

Figure 13-9 shows scenario 4, where one new table space and table were created (T12), some updates were done on the new table (T13), and there is a recovery object level (T14).



*Figure 13-9 Scenario 4*

Because no image copy was taken, DB2 recovers the table space by applying the logs, just as it did in previous versions, as shown in Example 13-26.

In DB2 9, there are three new values for column ICTYPE:

- ▶ “C”: When you create one new table space
- ▶ “E”: When you recovery (to current point)
- ▶ “M”: when you use MODIFY RECOVERY utility

*Example 13-26 Recovering output for the new table space created*

---

```

20.46 DSNUGUTC - RECOVER TABLESPACE DB2R7DB.DB2R7TS DSNUM ALL
:26:20.51 DSNUCALA - NO RECOVERY BASE AVAILABLE FOR RECOVERY OF
TABLESPACE DB2R7DB.DB2R7TS
:26:20.66 DSNUCALA - SYSLGRNX INFORMATION FOR MEMBER D9C1
:26:20.66 DSNUCALA - RECOVER UTILITY LOG APPLY RANGE IS RBA 00013531DCEC LRSN C4
RBA 00013531FEEE LRSN C4E70DC5B037
:26:20.66 DSNUCALA - RECOVER UTILITY LOG APPLY RANGE IS RBA 000135325847 LRSN C4
RBA 000000000000 LRSN C4E70F6CDBC1
20.67 DSNUCBLA - LOG APPLY PHASE COMPLETE, ELAPSED TIME = 00:00:00
20.68 DSNUCBDR - RECOVERY COMPLETE, ELAPSED TIME=00:00:00
20.68 DSNUGUTC - REBUILD INDEX(ALL) TABLESPACE DB2R7DB.DB2R7TS
:26:20.74 DSNUCRUL - UNLOAD PHASE STATISTICS - NUMBER OF RECORDS PROCESSED=6
20.74 DSNUCRIB - UNLOAD PHASE COMPLETE - ELAPSED TIME=00:00:00
:26:20.97 DSNURBXC - SORTBLD PHASE STATISTICS - NUMBER OF KEYS=6 FOR INDEX DB2R7
20.99 DSNUCRIB - SORTBLD PHASE STATISTICS. NUMBER OF INDEXES = 1
20.99 DSNUCRIB - SORTBLD PHASE COMPLETE, ELAPSED TIME = 00:00:00
21.04 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0

```

---

**Scenario 5**

Figure 13-10 shows scenario 5, where one BACKUP SYSTEM (T15) executes, and there is a RECOVERY object level using the LISTDEF with RI option (T16).

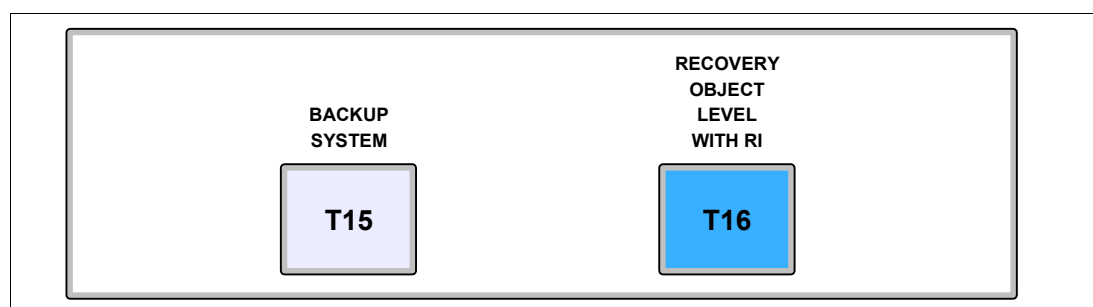


Figure 13-10 Scenario 5

Example 13-27 shows the new BACKUP SYSTEM FULL output.

*Example 13-27 BACKUP SYSTEM FULL output*

---

```
4:21:14.84 DSNUGUTC - BACKUP SYSTEM
14:21:15.39 DSNUVBBD - BACKUP SYSTEM UTILITY FOR DATA STARTING,
                  COPYPOOL = DSN$DB9C$DB
                  TOKEN = X'C4F9C3F1C4E6CB3BBD2B1A49C4E3145B2232'.
14:21:15.73 DSNUVBBD - BACKUP SYSTEM UTILITY FOR DATA COMPLETED SUCCESSFULLY,
                  COPYPOOL = DSN$DB9C$DB
                  TOKEN = X'C4F9C3F1C4E6CB3BBD2B1A49C4E3145B2232'
                  ELAPSED TIME = 00:00:00.
14:21:15.73 DSNUVBBD - BACKUP SYSTEM UTILITY FOR LOGS STARTING,
                  COPYPOOL = DSN$DB9C$LG
                  TOKEN = X'C4F9C3F1C4E6CB3BBD2B1A49C4E3145B2232'.
14:21:20.71 DSNUVBBD - BACKUP SYSTEM UTILITY FOR LOGS COMPLETED SUCCESSFULLY,
                  COPYPOOL = DSN$DB9C$LG
                  TOKEN = X'C4F9C3F1C4E6CB3BBD2B1A49C4E3145B2232'
                  ELAPSED TIME = 00:00:04.
14:21:20.71 DSNUVBBD - BACKUP SYSTEM UTILITY COMPLETED, ELAPSED TIME = 00:00:05.
14:21:20.71 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

---

Example 13-28 shows the output of a full recovery for a table space with referential integrity. With full recovery, because you are recovering the data fully, you can also recover just one object or a subset of objects from the RI set and the entire set will still be in sync.

The use of LISTDEF with RI option is specially recommended for point-in-time recovery. The system-level backup could be used to restore table space with referential integrity.

*Example 13-28 Backup full output and recovery for table space with referential integrity*

---

```
DSNUGUTC - LISTDEF MARCELO INCLUDE TABLESPACE DSN8D91A.DSN8S91D RI BASE
DSNUILDR - LISTDEF STATEMENT PROCESSED SUCCESSFULLY
DSNUGUTC - RECOVER LIST MARCELO
DSNUGULM - PROCESSING LIST ITEM: TABLESPACE DSN8D91A.DSN8S91D
DSNUGULM - PROCESSING LIST ITEM: TABLESPACE DSN8D91A.DSN8S91E
DSNUGULM - PROCESSING LIST ITEM: TABLESPACE DSN8D91A.DSN8S91P
DSNUCBMT - THE RECOVERY BASE FOR TABLESPACE DSN8D91A.DSN8S91D IS THE SYSTEM LEVEL BACKUP
42120, AND TOKEN X'C4F9C3F1C4E6CB3BBD2B1A49C4E3145B2232'
DSNUCBMT - TABLESPACE DSN8D91A.DSN8S91D WAS SUCCESSFULLY RESTORED FROM A FLASHCOPY

DSNUCBMT - THE RECOVERY BASE FOR TABLESPACE DSN8D91A.DSN8S91E IS THE SYSTEM LEVEL BACKUP
42120, AND TOKEN X'C4F9C3F1C4E6CB3BBD2B1A49C4E3145B2232'
DSNUCBMT - TABLESPACE DSN8D91A.DSN8S91E DSNUM 1 WAS SUCCESSFULLY RESTORED FROM A FLASHCOPY

DSNUCBMT - TABLESPACE DSN8D91A.DSN8S91E DSNUM 2 WAS SUCCESSFULLY RESTORED FROM A FLASHCOPY

DSNUCBMT - TABLESPACE DSN8D91A.DSN8S91E DSNUM 3 WAS SUCCESSFULLY RESTORED FROM A FLASHCOPY
```

---



## Scenario 6

Figure 13-11 shows scenario 6, where one new table is created (T17), some rows are inserted (T18), then one Incremental FlashCopy is taken (T19), and there is one ALTER column data type varchar(20) to char(20) (T20) and recovery object level (T21).

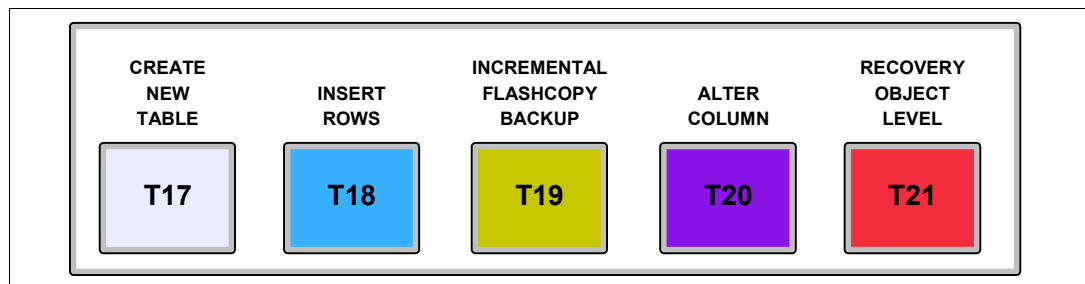


Figure 13-11 Scenario 6

Example 13-29 shows the output for the last incremental FlashCopy that was taken.

### Example 13-29 Last incremental FlashCopy output

```
28.14 DSNUGUTC - BACKUP SYSTEM ESTABLISH FCINCREMENTAL
28.67 DSNUVBBD - BACKUP SYSTEM UTILITY FOR DATA STARTING,
      COPYPOOL = DSN$DB9C$DB
      TOKEN = X'C4F9C3F1C4E8156B055314CBC4E3145B2232'.
29.89 DSNUVBBD - THE SYSTEM LEVEL BACKUP TAKEN IS AN INCREMENTAL FLASHCOPY
```

Example 13-30 shows as a sample output of using incremental FlashCopy for the recovery to a point prior to the ALTER COLUMN.

The DDL will not be modified to VARCHAR again. DB2 will manage this task using data versioning. Remember that when you did ALTER, the table space will stay in AREO (Advisory Reorg Status) status. The table space should be reorganized for optimal performance.

### Example 13-30 Recovery output

```
RECOVER TABLESPACE DB2R7DB.DB2R7TS DSNUM ALL
THE RECOVERY BASE FOR TABLESPACE DB2R7DB.DB2R7TS IS THE SYSTEM LEVEL BACKUP
TOKEN X'C4F9C3F1C4E8156B055314CBC4E3145B2232'
TABLESPACE DB2R7DB.DB2R7TS WAS SUCCESSFULLY RESTORED FROM A FLASHCOPY, ELAPSED

A - SYSLGRNX INFORMATION FOR MEMBER D9C1
A - RECOVER UTILITY LOG APPLY RANGE IS RBA 00013531DCEC LRSN C4E70DC5AE95 TO
      RBA 00013531FEEE LRSN C4E70DC5B037
A - RECOVER UTILITY LOG APPLY RANGE IS RBA 000135325847 LRSN C4E70DC60DBC TO
      RBA 00013533E2E0 LRSN C4E70F6CDE65
A - RECOVER UTILITY LOG APPLY RANGE IS RBA 00013549609E LRSN C4E814D632AB TO
      RBA 000000000000 LRSN C4E8164FF6B4
LOG APPLY PHASE COMPLETE, ELAPSED TIME = 00:00:00
RECOVERY COMPLETE, ELAPSED TIME=00:00:00
```

### Scenario 7

Figure 13-12 shows scenario 7, where we create a new index DB2R7.MARCELO2 (T22), insert some rows in the table(T23), and execute one recovery to point in time (T24).

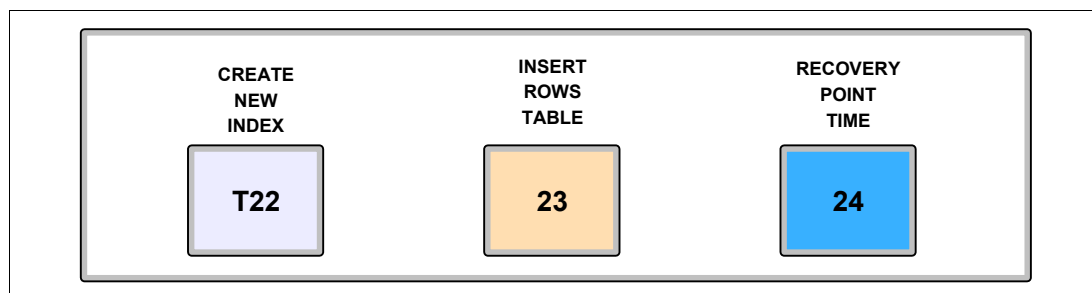


Figure 13-12 Scenario 7

Example 13-31 shows that the recovery of table space was done using a FlashCopy and all the indexes were in the rebuild pending status. If index (MARCELO2) had been dropped, the message would be the same. Recovery of the table space will occur and only index MARCELO1 will be in the rebuild pending status.

Example 13-31 Recovery index output

---

```

- RECOVER TABLESPACE DB2R7DB.DB2R7TS DSNUM ALL TOLOGPOINT X'C4E815715B84'
- THE RECOVERY BASE FOR TABLESPACE DB2R7DB.DB2R7TS  IS THE SYSTEM LEVEL BACKUP
  ND TOKEN X'C4F9C3F1C4E8156B055314CBC4E3145B2232'
- TABLESPACE DB2R7DB.DB2R7TS  WAS SUCCESSFULLY RESTORED FROM A FLASHCOPY, ELAP

CARS - INDEX DB2R7.MARCELO1 IS IN REBUILD PENDING
CARS - INDEX DB2R7.MARCELO2 IS IN REBUILD PENDING
CARS - ALL INDEXES OF DB2R7DB.DB2R7TS ARE IN REBUILD PENDING
CALA - SYSLGRNX INFORMATION FOR MEMBER D9C1
CALA - RECOVER UTILITY LOG APPLY RANGE IS RBA 00013531DCEC LRSN C4E70DC5AE95 TO
      RBA 00013531FEEE LRSN C4E70DC5B037
CALA - RECOVER UTILITY LOG APPLY RANGE IS RBA 000135325847 LRSN C4E70DC60DBC TO
      RBA 00013533E2E0 LRSN C4E70F6CDE65
CALA - RECOVER UTILITY LOG APPLY RANGE IS RBA 00013549609E LRSN C4E814D632AB TO
      RBA 000000000000 LRSN C4E815715B84
- LOG APPLY PHASE COMPLETE, ELAPSED TIME = 00:00:00
  
```

---

### Scenario 8

Figure 13-13 shows scenario 8, where we remove DASD volume SBOX5T from the database Copy Pool DSN\$DB9C\$DB(T25) and then try a recovery object level (T26).

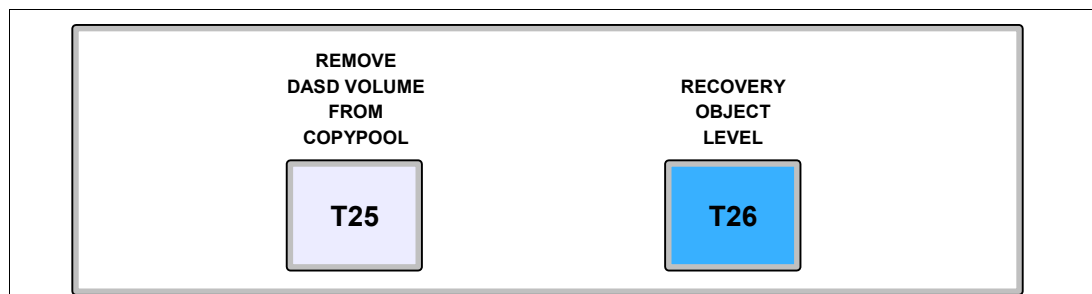


Figure 13-13 Scenario 8

Example 13-32 shows the DFSMSShsm storage group LISTVOL command listing for the SBOX5T DASD volume that is available on DFSMSShsm storage group DB9CDATA.

On third output shows the DB2 recovery job failing with RC=8 and the message ARC1172I with RC=1 on the z/OS log, stating that SMS cannot determine if the volume being selected is an SMS-managed volume, and the table space is set to recovery pending status.

Example 13-32 DFSMSShsm output

---

```
D SMS,SG(DB9CDATA),LISTVOL
IGD002I 14:02:17 DISPLAY SMS 544
```

STORGRP	TYPE	SYSTEM=	1	2	3	4	
DB9CDATA	POOL		+	+	+	+	

VOLUME	UNIT	SYSTEM=	1	2	3	4	STORGRP NAME
SBOX5S	DA02		+	+	+	+	DB9CDATA
<b>SBOX5T</b>	DB02		+	+	+	+	DB9CDATA
SBOX5U	DC02		+	+	+	+	DB9CDATA
SBOX5V	DD02		+	+	+	+	DB9CDATA

\*\*\*\*\* LEGEND \*\*\*\*\*

. THE STORAGE GROUP OR VOLUME IS NOT DEFINED TO THE SYSTEM

+ THE STORAGE GROUP OR VOLUME IS ENABLED

---

Example 13-33 shows the same command, but after SBOX5T has been removed from the DFSMSShsm storage group.

Example 13-33 DFSMSShsm output

---

```
-D SMS,SG(DB9CDATA),LISTVOL
IGD002I 15:06:13 DISPLAY SMS 735
```

STORGRP	TYPE	SYSTEM=	1	2	3	4	
DB9CDATA	POOL		+	+	+	+	

VOLUME	UNIT	SYSTEM=	1	2	3	4	STORGRP NAME
SBOX5S	DA02		+	+	+	+	DB9CDATA
SBOX5U	DC02		+	+	+	+	DB9CDATA
SBOX5V	DD02		+	+	+	+	DB9CDATA

\*\*\*\*\* LEGEND \*\*\*\*\*

. THE STORAGE GROUP OR VOLUME IS NOT DEFINED TO THE SYSTEM

+ THE STORAGE GROUP OR VOLUME IS ENABLED

---

Example 13-34 shows the DB2 recovery utility job failing with RC=8 and informing you that you need to use the MVS log to determine the correct DFSMSHsm error message.

*Example 13-34 DB2 recovery utility job*

---

```
11:47:12.21 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = DB2R7.DB2R7001
11:47:12.31 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
11:47:12.31 DSNUGUTC - RECOVER TABLESPACE DB2R7DB.DB2R7TS DSNUM ALL
11:47:12.35 DSNUCBMT - THE RECOVERY BASE FOR TABLESPACE DB2R7DB.DB2R7TS IS THE
SYSTEM LEVEL BACKUP
1008, TIME 145835, AND TOKEN X'C4F9C3F1C4E8156B055314CBC4E3145B2232'
11:47:12.38 DSNUCBMT - THE DFSMSHSM CALL TO RESTORE TABLESPACE DB2R7DB.DB2R7TS
FAILED WITH RC = X'00000008' AND REASON CODE = X'00000000'
SEE THE JOB LOG FOR DFSMSHSM MESSAGES INDICATING THE CAUSE OF THE ERROR
286 11:47:12.38 DSNUGSRX - TABLESPACE DB2R7DB.DB2R7TS IS IN RECOVER PENDING
11:47:12.38 DSNUCBDR - RECOVERY COMPLETE, ELAPSED TIME=00:00:00
11:47:12.39 DSNUGBAC - UTILITY EXECUTION TERMINATED, HIGHEST RETURN CODE=8
```

---

Example 13-35 shows message ARC1172I with RC=11 in the z/OS log, showing that SMS cannot determine if the volume being selected is an SMS-managed volume, and the table space is set to the recovery pending status.

*Example 13-35 Output for SMS command and fail recovery job*

---

```
ARC1860I THE FOLLOWING 0001 DATA SET(S) FAILED DURING 879
ARC1860I (CONT.) FAST REPLICATION DATA SET RECOVERY:
ARC1860I (CONT.) DB9CD.DSNDBC.DB2R7DB.DB2R7TS.I0001.A001, COPYPOOL=DSN
$DB9C$DB, DEVTYPE=DASD, VOLUME=SB0X5T, ARC1172I, RC=0011
```

---

## Scenario 9

Figure 13-14 shows scenario 9, where we run BACKUP SYSTEM DUMPONLY for a specific token (T27).

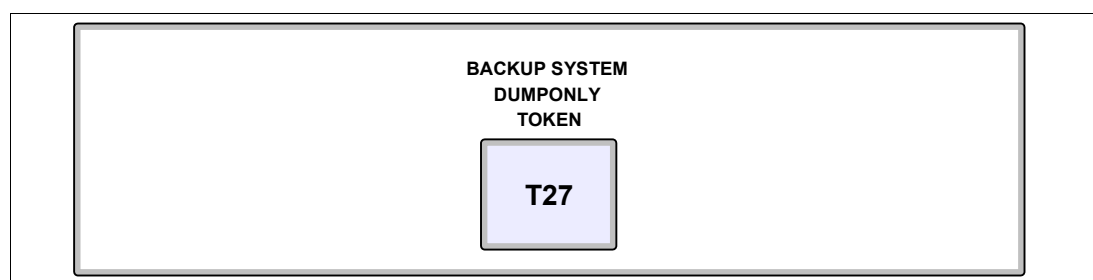


Figure 13-14 Scenario 9

Example 13-36 shows the output for the system-level backup with the specific token that was taken previously.

*Example 13-36 Output for DUMPONLY*

---

```
20:50:58.51 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = BKSYS
20:50:58.59 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
20:50:58.59 DSNUGUTC - BACKUP SYSTEM DUMPONLY
TOKEN=X'C4F9C3F3C4F29B86702DCC96C4EE5C1707BD'
20:50:58.73 DSNUVBBD - BACKUP SYSTEM UTILITY FOR DATA STARTING,
    COPYPOOL = DSN$DB9C$DB
    TOKEN = X'C4F9C3F3C4F29B86702DCC96C4EE5C1707BD'.
20:50:58.74 DSNUVBBD - BACKUP SYSTEM UTILITY FOR DATA COMPLETED SUCCESSFULLY,
    COPYPOOL = DSN$DB9C$DB
    TOKEN = X'C4F9C3F3C4F29B86702DCC96C4EE5C1707BD'
    ELAPSED TIME = 00:00:00.
20:50:58.74 DSNUVBBD - BACKUP SYSTEM UTILITY FOR LOGS STARTING,
    COPYPOOL = DSN$DB9C$LG
    TOKEN = X'C4F9C3F3C4F29B86702DCC96C4EE5C1707BD'.
20:50:58.74 DSNUVBBD - BACKUP SYSTEM UTILITY FOR LOGS COMPLETED SUCCESSFULLY,
    COPYPOOL = DSN$DB9C$LG
    TOKEN = X'C4F9C3F3C4F29B86702DCC96C4EE5C1707BD'
    ELAPSED TIME = 00:00:00.
20:50:58.74 DSNUVBBD - BACKUP SYSTEM UTILITY COMPLETED, ELAPSED TIME = 00:00:00.
```

---

The DB2 BACKUP SYSTEM DUMPONLY job returns with RC=0, but you have to check elsewhere to see if HSM did its part. An option is shown in Example 13-37, where you can look at the MVS log to discover HSM messages related to this job.

*Example 13-37 HSM messages on the MVS log*

---

```
*IEC501A M 0409,PRIVAT,SL,COMP,DFHSM70,DFHSM70,HSM.DMP.DB9CTST2.VSBOX5S
.D09289.T425123
IEC705I TAPE ON 0409,VT0014,SL,COMP,DFHSM70,DFHSM70,HSM.DMP.DB9CTST2.V
SBOX5S.D09289.T425123,MEDIA5
*IEC501A M 0409,PRIVAT,SL,COMP,DFHSM70,DFHSM70,HSM.DMP.DB9CTST2.VSBOX5S
.D09289.T425123
IEC705I TAPE ON 0409,VT0015,SL,COMP,DFHSM70,DFHSM70,HSM.DMP.DB9CTST2.V
SBOX5S.D09289.T425123,MEDIA5
```

---

A second option is to run the job shown in Example 13-38 to LIST the Copy Pool and check if the work was done correctly.

*Example 13-38 LIST COPYPOOL job*

---

```
//STEP01 EXEC PGM=IKJEFT01
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSTSIN DD *
    HSEND CMD WAIT LIST COPYPOOL(DSN$DB9C$DB) -
        ODS(DB2R7.TEST2) -
        ALLVOLS(TOKEN(X'C4F9C3F3C4F29B86702DCC96C4EE5C1707BD'))
```

---

Example 13-39 shows the output for the LIST Copy Pool, where you can see the columns DUMPSTATE, SOURCE, and DUMPVOLS with the information about the DASD volume source and tapes allocated for this dump.

*Example 13-39 Output LIST COPYPOOL*

---

```

COPYPOOL=DSN$DB9C$DB
ALLOWPPRCP FRB=NO FRR=NO

VERSION  VTOCENQ    DATE          TIME          FASTREPLICATIONSTATE  DUMPSTATE
   057      N      2009/10/16      23:51:42      NONE                ALLCOMPLETE
TOKEN(C)=C'D9C3D29f0.öoD0*..''
TOKEN(H)=X'C4F9C3F3C4F29B86702DCC96C4EE5C1707BD'
TOTAL NUM OF VOLUMES=00004,INCREMENTAL=N

SGNAME    SOURCE - TARGET  SOURCE - TARGET  SOURCE - TARGET  SOURCE - TARGET
DB9CDATA  SBOX5S - *****  SBOX5T - *****  SBOX5U - *****  SBOX5V - *****

DUMPCCLASS  REQUIRED  DUMPSTATE  VOLSSUC    EXPDATE    AVAILABLE
DB9CTST2    Y        COMPLETE  00004      2009/10/17  Y

                HWCOMP  ENCRYPT  ENCTYPE  RSAKEY/KPWD
                NO      NONE    *****  *****

SOURCE  DUMPVOLS                                     DEVICE TYPE
SBOX5S  VT0014 VT0015 VT0016 VT0017 VT0018 VT0019      3590-1
FILE SEQ=01, DSNAME=HSM.DMP.DB9CTST2.VSBOX5S.D09289.T425123
SBOX5T  VT0019 VT0020 VT0021 VT0026 VT0027 VT0028      3590-1
        VT0029      *****
FILE SEQ=02, DSNAME=HSM.DMP.DB9CTST2.VSBOX5T.D09289.T425123
SBOX5U  VT0029 VT0030 VT0031 VT0033 VT0034 VT0035      3590-1
FILE SEQ=03, DSNAME=HSM.DMP.DB9CTST2.VSBOX5U.D09289.T425123
SBOX5V  VT0035 VT0036 VT0037 VT0038 VT0057      3590-1
FILE SEQ=04, DSNAME=HSM.DMP.DB9CTST2.VSBOX5V.D09289.T425123

----- END OF -- COPY POOL -- LISTING -----

```

---

**Tip:** Run the list Copy Pool job to discover if your dump executed correctly. If you find some problems for different DB2s, than you have to go to the MVS log to discover the HSM message.

## 13.10 Considerations for using BACKUP and RESTORE SYSTEM

### **Advantages**

The advantages are:

- ▶ Suitable for ERP applications and Disaster Recovery.
- ▶ A full DB2 system backup is logically completed in seconds.
- ▶ Object-level recovery works fine when you want to recover a specific object.

- ▶ A BACKUP SYSTEM or INCREMENTAL BACKUP job is much simpler because you do not need to specify all the table spaces and index spaces. This is especially important in an SAP environment where the list of table spaces is constantly changed by SAP and the list is very large.
- ▶ Indexes do not need to be rebuilt when you use SYSTEM RESTORE.
- ▶ The quickness of the backup allows that copies can be made more frequently. In the event of a failure, the amount of lost data might be reduced. For example, you could run BACKUP SYSTEM for your critical application more frequently. This would reduce the recovery time for specific objects because there is less to roll forward.
- ▶ With Incremental system backup, you can minimize the I/O impact of the background physical copy.
- ▶ The elapsed time of the physical copy is reduced considerably.

### ***Disadvantages***

The disadvantages are:

- ▶ The inability to RECOVER at the object level using a system-level backup if an object has changed DASD volumes since the backup was created.
- ▶ There will be situations where RESTORE may fail, such as:
  - You remove any volume from the Copy Pool.
  - You initialize any volume that belongs to the Copy Pool after a backup is taken.
  - You have data sets that have moved volumes since a backup was taken.
- ▶ Incremental FlashCopy to DASD is not suitable for point-in-time recovery, as it will always keep only the last version of the updated data.  
If you create a dump of the incremental FlashCopy to tape, then you will have preserved the point-in-time copy for recovery.
- ▶ When the backup system or restore system fails, it is difficult to find and understand the messages in HSM.
- ▶ It is a challenge to administer because either the DB2 system team has to understand SMS and HSM or the storage team needs to understand DB2.
- ▶ For installations that use Extended Remote Copy (XRC), now known as z/OS Global Mirror, the RESTORE SYSTEM or the RECOVER utility cannot use FlashCopy to restore the entire DB2 system from a Copy Pool backup:
  - As an alternative to solving this problem, BACKUP SYSTEM on tape could be used for RESTORE SYSTEM or object level recovery. This option is slow but is easier.

**Note:** In DB2 9, the BACKUP SYSTEM utility will update the COPYLASTIME timestamp for all objects in the system.

- Even though this alternative will make the restore faster than the previous alternative, it requires much work on the part of the storage team, who must disable z/OS Global Mirror and reenablen it afterwards.

### **Recommendations**

We make the following recommendations for the BACKUP and RESTORE SYSTEM utility:

- ▶ Create a process in charge of coordination between DB2 systems programmers, storage administrators, and DBA teams when they create new tables in a new DASD, remove a DASD from the Copy Pool, or move a table space from one DASD to another.
- ▶ If you are planning to use the BACKUP SYSTEM utility, you have to have a dedicated DASD for each DB2 subsystem. If you have table spaces from more than one DB2 in the same DASD volume, when you execute a RESTORE BACKUP SYSTEM for a specific DB2 subsystem, you could have inconsistent data restored.

## **13.11 Other available improvements**

Other available improvements for the BACKUP and RESTORE SYSTEM utility are:

- ▶ BACKUP SYSTEM FULL fails if any of the source volumes are in an existing FlashCopy relationship. This can occur when using the CHECK utility SHRLEVEL(CHANGE), which creates the shadow data set in the same storage group as the production page set.

To solve this problem, implement the new DSNZPARM parameter UTIL\_TEMP\_STORCLAS provided by PK41711 (UK41370). This new parameter will allow customers to specify a storage class that points to storage group with non-PPRC volumes.

- ▶ With z/OS V1.8 APAR OA23489 (PTF UA42371) or z/OS V1.9 (PTF UA42372), BACKUP SYSTEM and RESTORE SYSTEM can use FlashCopy to back up and restore data to and from DASD volumes that are PPRC primaries. APAR OA24814 is also required.

## **13.12 RESTORE SYSTEM**

The RESTORE SYSTEM utility is used to perform a point-in-time recovery (PITR). After the log truncation point is established in the BSDS, RESTORE SYSTEM restores the appropriate copy of the database and log Copy Pools to bring the DB2 system up to the log truncation point.

DSNJU003 is used to establish the log truncation point through the CRESTART statement. The SYSPITR option is used when providing a RBA or a LRSN for the log truncation point. SYSPITRT is used when providing a date and time of day as the log truncation point. All members of a data sharing system should establish the log truncation point.

**Attention:** If DB2 data sharing has been enabled, and some of the members are passive and used as secondary members in support of data base server failover, it is important to run an ARCHIVE LOG SCOPE(GROUP) prior to bringing the DB2 group down for recovery. This ensures that all of the DB2 members have written out the most current LRSN, and the recover to time is not limited by members with low to no activity. This is also true when using date and time as the log truncation point.

The RESTORE SYSTEM utility does not restore logs. The utility only applies the active logs.



## 13.13 Backing up the log Copy Pool prior to system-level recovery

When preparing to perform a PITR, we recommend, as a best practice, that you back up the DSN\$location\_name\$LG Copy Pool prior to establishing any log truncation points. When performing a system-level PITR, the DSN\$location\_name\$LG Copy Pool is not restored, in contrast to the DSN\$location\_name\$DB Copy Pool. After the log truncation has been established and the DB2 system conditionally restarted, the DSN\$location\_name\$DB Copy Pool is restored, during the execution of the Restore System utility, to its status at the time of a prior Backup System utility copy. It is a best practice to ensure that the DB2 system can always be taken back to the way it looked before any recovery actions. It is only necessary to back up the DSN\$location\_name\$LG Copy Pool in order to provide a way back. This provides the ability to correct or change a log truncation point and redo the PITR. This includes another arbitrary point in time or to current point in time.

The DFSMSHsm FRBACKUP command is used to provide the backup for the log Copy Pool.

Example 13-40 shows how to code the FRBACKUP command. To execute this command, you need to bring down the DB2 system.

*Example 13-40 FRBACKUP JCL*

---

```
//STEP01 EXEC PGM=IKJEFT01
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSTSIN DD *
        HSEND FRBACKUP COPYPOOL(DSN$DB9C$LG) EXECUTE
```

---

The output is shown in Example 13-41.

*Example 13-41 DFSMSHsm output*

---

```
ARC1805I THE FOLLOWING 00004 VOLUME(S) WERE 558
ARC1805I (CONT.) SUCCESSFULLY PROCESSED BY FAST REPLICATION BACKUP OF
ARC1805I (CONT.) COPY POOL DSN$DB9C$LG
ARC1805I (CONT.) SBOX5Q
ARC1805I (CONT.) SBOX5R
ARC1805I (CONT.) SBOX5A
ARC1805I (CONT.) SBOX5B
ARC1802I FAST REPLICATION BACKUP HAS COMPLETED FOR 577
ARC1802I (CONT.) COPY POOL DSN$DB9C$LG, AT 19:57:38 ON 2009/10/22,
ARC1802I (CONT.) FUNCTION RC=0000, MAXIMUM VOLUME RC=0000
```

---

Example 13-42 shows that the copy created is the current copy on DASD.

*Example 13-42 DFSMSHsm list report*

---

```
COPYPOOL=DSN$DB9C$LG
ALLOWPPRCP FRB=NO FRR=NO

VERSION  VTOCENQ    DATE          TIME          FASTREPLICATIONSTATE  DUMPSTATE
   055      Y      2009/10/22      19:57:29      RECOVERABLE          NONE
TOKEN(C)=C''
TOKEN(H)=X''
TOTAL NUM OF VOLUMES=00018,INCREMENTAL=N

SGNAME    SOURCE - TARGET  SOURCE - TARGET  SOURCE - TARGET  SOURCE - TARGET
DB9CARCH  SBOX5Q - SBOX6G  SBOX5R - SBOX6H
DB9CLOG1  SBOX5A - SBOX6I  SBOX5B - SBOX6J
```

---

For more informations about RESTORE SYSTEM, refer to:

- ▶ *Disaster Recovery with DB2 UDB for z/OS*, SG24-6370
- ▶ *Optimizing Restore and Recovery Solutions with DB2 Recovery Expert for z/OS V2.1*, SG24-7606
- ▶ *DB2 9 for z/OS: Backup and Recovery I/O Related Performance Considerations*, REDP-4452

## 13.14 Conditional restart after restoring the log Copy Pool

If you restore the log Copy Pool, then a conditional restart, using the data complete LRSN (of the system-level backup) as the ENDLRSN, ENDRBA, or SYSPITR, is required. This avoids using the part of the DB2 logs that is “fuzzy”, because a system-level backup is a fuzzy, rolling copy.

The BACKUP SYSTEM utility takes fuzzy, rolling backups of the database and log Copy Pools because the FlashCopy is done at the disk volume level and not as a FlashCopy consistency group. If the backup of the log Copy Pool is used for a cloned system or is restored via DFSMSHsm for a system point-in-time recovery, then a conditional restart of DB2 should be done instead of a normal restart for these two cases:

- ▶ A DB2 system where the active log data sets are striped.
- ▶ A DB2 data sharing system.

When active log data sets are striped, the striping is done at the control interval (CI) level and two adjacent CIs could be striped to two different disk volumes. During the backup of the log Copy Pool, because the FlashCopy is done at the disk volume level, a more recent CI may be written to the FlashCopy target volume before an older CI.

In a data sharing environment, each member has its own active log data sets that reside on different disk volumes. Depending on the order in which the volumes are FlashCopied during the BACKUP SYSTEM utility, the backup of the log Copy Pool may contain log records from some members but may be missing log records from other members, during the period of the backup.

With APAR PK89645, the DSNU1614I message issued by the BACKUP SYSTEM utility is modified to externalize the data complete LRSN. The data complete LRSN is the RBA or LRSN of the last log record written to the active log data set after the database Copy Pool has been backed up and before the log Copy Pool has been backed up. For cloned systems, or for system point-in-time recovery where the log copy pool is restored, the data complete LRSN should be used as the log truncation point value for conditional restart.

For disaster recovery procedures where the system-level backup (SLB) is shipped on tape to the remote site, if data sharing or if the active log data sets were striped at the production site, then after the log Copy Pool and database Copy Pool have been restored at the remote site (outside of DB2), a conditional restart of DB2 should be done using the data complete LRSN as the log truncation point (ENDRBA or ENDLRSN).

To ensure a successful restart when the active log data sets are striped, or to ensure, in a data sharing environment, that the system reflects all updates at a specific log point, you should, for a DB2 cloned system, do a conditional restart using an ENDRBA or ENDLRSN that is equal to the data complete LRSN of the SLB.

For a DB2 system being recovered to a point in time (system point-in-time recovery) where the log copy pool has been restored from a SLB, a conditional restart should be done with a log truncation point equal to or less than the data complete LRSN of the SLB. For example, use the data complete LRSN as the CRESTART ENDRBA, ENDLRSN, or SYSPITR log truncation point.





## Operational considerations

In this chapter, we focus on the various operational considerations when running utility jobs. The DB2 utilities are used for routine object maintenance and are often run on a frequent basis in many shops. In a 24x7 world, batch windows are getting shorter and shorter and in some companies they are non-existent. The time it takes for a utility to complete is a concern. We want to ensure utilities run as quickly and nondisruptively as possible.

One way is to avoid having to run the utility at all. Of course, the best utility is the one you do not need to run, and there are new ways to help you decide when utilities should be run. DB2 also provides more and more online utilities that allow for concurrent access while executing.

The more applications that can make use of isolation level uncommitted read, the higher the availability to the application and the more likely utilities are nondisruptive.

You may need to be concerned about utility execution and performance, how to determine in which phase a utility is running, avoiding contention with other utilities, what to do when a utility fails, and other aspects from an operational point of view.

The DB2 utilities, except for the stand-alone utilities, run under the control of DB2 and therefore provide the highest degree of data integrity. But with this functionality comes a hidden cost and longer path length, that is, DB2 uses buffer pools for caching the objects data pages. As a result, there can be some buffer pool improvements that favor running utilities. We cover these considerations as well from a high-level point of view.

For I/O specific considerations, refer to *DB2 9 for z/OS: Backup and Recovery I/O Related Performance Considerations*, REDP-4452.

The following topics are presented:

- ▶ Utility execution
- ▶ Parallelism
- ▶ Concurrency
- ▶ Data sharing considerations
- ▶ Buffer pool considerations
- ▶ Specific utility considerations
- ▶ Miscellaneous considerations
- ▶ DSNZPARMs that affect utility execution

## 14.1 Utility execution

You need to prepare the utility job for execution and maximize system resources so that the job runs smoothly and utilizes the minimum amount of system resources. It is important to understand what to be concerned with when setting up the utility job. In view of these concerns, this section includes the following topics:

- ▶ Preparing for utility execution
- ▶ During utility execution
- ▶ Compatibility of utilities in restrictive states
- ▶ Stopped utilities
- ▶ Restarting utilities

### 14.1.1 Preparing for utility execution

We discuss considerations when preparing your utility job for execution in this section:

- ▶ Memory considerations
- ▶ Region size
- ▶ Sort work data sets
- ▶ Data set allocations

#### **Memory considerations**

Virtual memory should be backed by real storage to avoid paging. If virtual storage is constrained, then this will have an impact on the degree of parallelism chosen by DB2 and also on sort processing.

#### ***Region size***

The region size should reflect the maximum amount of private storage to be allocated to the job. Allocating more storage allows DB2 to choose the highest degree of parallelism within the storage available.

The REGION parameter specified in JCL or a JES parameter results in two values that are used for virtual storage management: region size value and limit value.

- ▶ Region size

The region size determines the amount of storage that can be allocated to a job or step for any single GETMAIN.

- ▶ Limit value

The limit value is the maximum total storage that a job or step can allocate.

#### ***User exits***

Two user exits can influence the setting of these two values: IEALIMIT and IEFUSI. IEFUSI takes precedence over IEALIMIT. IEALIMIT has no control over the amount of virtual storage that can be allocated above the 16 MB line.

#### ***What region size is recommended***

The often mentioned recommended region size for a batch job is 0 M. Ideally, this will help ensure that the utility job will have the best degree of parallelism possible. However, this is not always feasible in a production environment. The stress on real memory is too high at some sites, especially if you are running multiple utility jobs in parallel.

If every utility batch job were to ask for 0 M, it is possible these jobs might be constrained because of the lack of enough real memory to back virtual storage. You do not want to introduce system wide paging at the expense of specifying too much storage. You really need to carefully evaluate an appropriate region size based on your workload constraints.

### Region size explained

The region size allocated to your batch job depends on a number of factors. One important factor is the z/OS IEFUSI exit. If you install the exit, and limit the amount of memory therein, then the value you specify in the exit takes precedence. Here we consider what happens to the private storage both below and above the 16 MB line based on the specific value you specify in the REGION parameter.

In Table 14-1, we summarize the actual storage allocated to your batch job when you specify the REGION parameter in the JCL with different values. This assumes you have not installed the IEFUSI exit to limit the amount of memory to be allocated to a batch job.

Table 14-1 Storage allocated using REGION parameter: No IEFUSI exit installed

REGION=value	Below 16 MB line Private area	Above 16 MB line Extended private area
value=0M	Use all available space,	Use all available space.
0<value<16M	Use requested space. If it is not available, terminate the job.	Use default value of 32M.
16M<value<32M	Use all available space.	Use 32 MB.
value>32M	Use all available space.	Use specified value. If it is not available, terminate the job.

### Sort work data sets

To maximize the performance of any utility that uses sort, we recommend that you set up the utility job for dynamic allocation of sort work data sets. This is achieved by using the SORTNUM elimination feature, which is discussed in Chapter 6, “Sort processing” on page 141. This is the preferred method for usability as well as performance.

However, ensure that the storage class for these dynamically allocated sort work data sets have enough volumes assigned to it, or the utility job may fail due to insufficient sort work space. Often, customers have storage classes set up using SMS routines where specific hlq map to certain storage classes. Dynamically allocated data sets may be routed to a storage pool where there may not be enough volumes to satisfy multiple concurrent utility jobs requiring sort space.

### Data set allocations

Space management is a huge concern for many DB2 sites. Often, in the past, you had to micro-manage your DB2 data set allocation, in an attempt to avoid too many extents. Having a number of extents used to cause performance issues. With the advent of new DASD technology, this is no longer a performance concern, but more of an availability issue. You do not want to get too close to the maximum number of extents because you may need an application outage to correct the situation. These days, many sites are opting for either SMS-managed or DB2-managed rather than the cumbersome user-managed data sets paradigm.

Another reason to go with SMS-managed or DB2-managed rather than user-managed is that user-managed does not allow for exploitation of some of the new disk technological features, such as extended format data sets, which allows DFSMS striping and a DSSIZE greater than 4 GB.

**Note:** SAP does not support user-managed data.

SMS-managed and DB2-managed provide a higher level of automation in space management than user-managed and is generally preferred. For more information about these choices, refer to the *DB2 for z/OS Version 9.1 Administration Guide*, SC18-9840.

There are a number of considerations to evaluate for different data set allocations when running utility jobs:

- ▶ Are you using SMS-managed, user-managed, or DB2 managed data sets?
- ▶ Are you using the sliding scale option to allow DB2 to automatically choose an appropriate secondary extent space value when a linear data set needs an extent?
- ▶ What recommendations are there when choosing utility work data sets or tape data sets?
- ▶ When do you need to allocate a shadow data set or a mapping table?

These are all addressed in this section based on these categories:

- ▶ Linear data sets
- ▶ Utility work data sets
- ▶ Tape data sets
- ▶ Shadow data sets
- ▶ Mapping table for REORG SHRLEVEL CHANGE

### ***Linear data sets***

Recent synergy between DB2 and SMS allow the two products to be better integrated. DB2 has the ability to include SMS-managed attributes for storage groups. This includes using DATACLAS, MGMTCLAS, and STORCLAS on the CREATE and ALTER STOGROUP commands.

In addition, many sites now use the DB2 space allocations sliding scale for secondary extents to minimize the problem of running out of extents. This is enabled by setting the DSNZPARM MGEXTSZ to YES and an objects SECQTY to -1. DB2 will then manage secondary extent sizes and determine an appropriate secondary space value when an extent is to be allocated. This feature tries to provide for the best utility performance by minimizing extent processing. For more information about DSNZPARM MGEXTSZ, refer to Table 14-8 on page 402.

Use the following recommendations for linear data sets:

- ▶ Allocate sufficient space in cylinders  
Set PRIQTY and SECQTY to avoid extent processing, particularly for LOAD, REORG, or REBUILD INDEX.
- ▶ If secondary extents are unavoidable during utility processing, set DSNZPARM MGEXTSZ to YES to allow DB2 to manage secondary extent allocation and use a high SECQTY or -1.
- ▶ Use DS8000® DASD for best performance and throughput.
- ▶ Use the VARY DS CONTROL INTERVAL parameter set to YES on installation window DSNTIP7 to allow DB2-managed data sets to have variable VSAM control intervals corresponding to the size of the buffer pool that is used for the table space or index space (page size).



- Use DB2 and SMS managed data sets  
Use SMS managed storage groups with a volume ID of \* instead of the actual VOLSERs.

### **Utility work data sets**

There are a number of considerations for the utility work data sets. They are discussed in more detail in Chapter 6, “Sort processing” on page 141.

The following recommendations are discussed in this section:

- Define ACS routines for data set allocation.
- Stripe SYSREC, SYSUT1, and SORTOUT.
- Ensure correct SORTWORK data set placement.
- Prevent OEM override of space allocation.
- Ensure sufficient free space in the sort work pool.

### **Define ACS routines for data set allocation**

The ACS routines can use different attributes in a job as variables. ACS routines can examine many SMS read-only variables, such as job name, RACF group, LPAR name, Sysplex name, user name, job/step name, and accounting information. These SMS read-only variables are listed in a technote entitled “DFSMSrmm SMS ACS Support” at:

<http://www.redbooks.ibm.com/abstracts/tips0530.html?Open>

Use ACS routines to define policies for enforcing the data set naming standards and to control data set placement, device type, volume selection, and space usage for both DASD and TAPE data sets for SYSUT1, SORTOUT, and other data sets. Set up ACS routines to exclude SORTWK\*, STATWK\*, DATAWK\*, DAnnWK\*, STnnWK\*, and WnnWK\* data sets from VIO.

### **Stripe SYSREC, SYSUT1, and SORTOUT**

The use of DFSMS striping can provide a significant reduction in elapsed time for certain utility processing. See Table 14-2 for recommendations about which ddnames can benefit from DFSMS striping when you are dealing with larger objects and certain utilities.

*Table 14-2 Utilities that can benefit from DFSMS striping*

Utility	DDNAMEs for striping
LOAD	SYSREC, SYSUT1, and SORTOUT
REORG INDEX	SYSUT1
REORG TABLESPACE	SYSREC, SYSUT1, and SORTOUT

### **Ensure correct SORTWORK data set placement**

Ensure that sort work allocations are directed to those devices with the fastest available random I/O service times. Spread I/O for sort work files by assigning logical volumes on different physical devices to avoid cache and channel contention. In general sort work EXCPs are large, which could have a noticeable impact on measured disconnect times for other applications, hence isolate sort work devices particularly from data with critical response time requirements.

### **Prevent OEM override of space allocation**

Exclude SORTWK\*, STATWK\*, DATAWK\*, DAnnWK\*, STnnWK\* and SWnnWK\* data sets from any OEM products that reduce space allocation.

### ***Ensure sufficient free space in sort work pool***

As a general rule, the work pool should average over 20% free to allow for optimal allocation of sort work data sets and reduce the risk of failures due to lack of work space (messages ICE083A, ICE046A, etc.). If the percent free of the work pool is low, increase the number of volumes in the work pool or reduce the number of concurrently running utilities.

### ***Tape data sets***

DB2 9 supports 256K block sizes for tapes. To enable the use of 256K block sizes, change the DEVSUPxx member of SYS1.PARMLIB by setting TAPEBLKSZLIM to 262144. This setting can also be dynamically activated in SDSF by entering the command T DEVSUP=xx.

The ideal situation, if possible, is to utilize a high capacity drive for best tape performance. If you are using a single tape drive, choose the B-side, which gives approximately 7% higher throughput than the A-side.

### ***Shadow data sets***

The following utilities need shadow data sets allocated before the utility executes:

- ▶ REORG with SHRLEVEL REFERENCE or CHANGE
- ▶ CHECK INDEX with SHRLEVEL CHANGE
- ▶ CHECK DATA with SHRLEVEL CHANGE
- ▶ CHECK LOB with SHRLEVEL CHANGE

When using user-managed data sets, you need to preallocate the necessary shadow data sets prior to running any of these utilities. For SMS-managed or DB2-managed data sets, this process is simple: DB2 will automatically create the necessary shadow data sets and delete them when the utility successfully terminates.

When you preallocate the shadow data sets, keep in mind the following rules for user-managed data sets:

- ▶ Define the shadow data sets as LINEAR.
- ▶ Use SHAREOPTIONS(3,3).
- ▶ Define the shadow data sets as EA-enabled if the original table space or index space is EA-enabled.
- ▶ Allocate the shadow data sets on the volumes that are defined in the storage group for the original table space or index space.
- ▶ If you specify a secondary space quantity, DB2 does not use it. Instead, DB2 uses the SECQTY value for the table space or index space.

A recommended method to use to preallocate the shadow data sets is to use the MODEL parameter, which allows the shadow data set to be defined with the same attributes as the original. Example 14-1 demonstrates this method.

*Example 14-1 DEFINE CLUSTER using MODEL parameter*

---

```
DEFINE CLUSTER +
  (NAME('catname.DSNDBC.dbname.psname.x0001.L001') +
  MODEL('catname.DSNDBC.dbname.psname.y0001.L001')) +
  DATA
  (NAME('catname.DSNDBD.dbname.psname.x0001.L001') +
  MODEL('catname.DSNDBD.dbname.psname.y0001.L001'))
```

---

Here the instance qualifiers x and y are distinct and are equal to either I or J. You must determine the correct instance qualifier to use for a shadow data set by querying the DB2 catalog for the database and table space. An example of such a query for a table space and an index space are provided in Example 14-2 and Example 14-3, respectively.

*Example 14-2 Query for instance qualifier for a table space*

---

```
SELECT DBNAME, TSNAME, IPREFIX
  FROM SYSIBM.SYSTABLEPART
 WHERE DBNAME = 'dbname'
    AND TSNAME = 'psname';
```

---

*Example 14-3 Query for instance qualifier for an index space*

---

```
SELECT DBNAME, IXNAME, IPREFIX
  FROM SYSIBM.SYSINDEXES X, SYSIBM.SYSINDEXPART Y
 WHERE X.NAME = Y.IXNAME
    AND X.CREATOR = Y.IXCREATOR
    AND X.DBNAME = 'dbname'
    AND X.INDEXSPACE = 'psname';
```

---

A REORG SHRLEVEL CHANGE PART action is more complicated. Here Lnnn represents a partition identifier. The possible values for a partition identifier depend solely on the partition number, and are summarized in Table 14-3.

*Table 14-3 Mapping of partition identifiers and partition numbers*

Partition identifier	Partition number
A001-A999	1-999
B000-B999	1000-1999
C000-C999	2000-2999
D000-D999	3000-3999
E000-E996	4000-4096

If you have a partitioned table space, you want to determine the proper naming convention for the data sets in order to successfully execute the REORG utility with the SHRLEVEL CHANGE PART n:m parameter.

When reorganizing a partition, you must create the shadow data sets for the partition of the table space and for the partition of the partitioning index. In addition, you must create a shadow data set for each non-partitioning index that resides in user-defined data sets.

The name for this shadow data set has the form *catname.DSNDBx.dbname.psname.y0mmm.lnnn*. The variables are described in Table 14-4.

Table 14-4 Data set name identifier and value

Data set name identifier	Value
catname	VSAM catalog name (VCAT) or alias
x	C = CLUSTER component D = DATA component
dbname	Database name
psname	Pageset name
y	I or J
mmm	001 or 002
lnnn	Partition identifier - see Table 14-3

The query of the DB2 catalog table SYSTABLEPART (Example 14-4) provides the proper prefix for the shadow data sets of the partitioned table.

Example 14-4 Query against SYSTABLEPART

---

```
SELECT DBNAME, TSNAME, PARTITION, IPREFIX FROM SYSIBM.SYSTABLEPART
WHERE DBNAME = 'dbname' AND TSNAME = 'tablespacename'
ORDER BY PARTITION;
```

---

The query of the DB2 catalog table SYSINDEXPART in Example 14-5 provides the proper prefix for the shadow data sets of the partitioned index.

Example 14-5 Query against SYSINDEXPART

---

```
SELECT IXNAME, PARTITION, IPREFIX FROM SYSIBM.SYSINDEXPART
WHERE IXNAME = 'indexname'
ORDER BY PARTITION;
```

---

### **Mapping table for REORG SHRLEVEL CHANGE**

REORG SHRLEVEL CHANGE requires a mapping table and a unique index to be allocated prior to running the utility. During the REORG, rows are loaded into the shadow table space and each RID potentially changes from its original location. The sole purpose of the mapping table is so that DB2 can keep track of the original RIDs of data records and match it to the RIDs in the shadow copy. The mapping table holds an entry for each row in the table space with a one-to-one correspondence between the original RID and the new RID.

For more information, refer to Chapter 8, “Reorganizing data” on page 205.

## **14.1.2 During utility execution**

As your utility executes, there are a number of areas allow you to monitor its execution. For example, you might be running a critical RECOVER utility and you want to monitor where in the process it is so that you might determine approximately when it will complete.

When monitoring, you should be aware of the following tools:

- ▶ Utility messages
- ▶ Claims and drains
- ▶ UTSERIAL lock

## Utility messages

You can view the utility and see the status of the utility by using the `-DISPLAY UTILITY` command. This view gives you an idea of the utility's progress.

The output shown in Example 14-6 is the result of running a REORG TABLESPACE utility job. You can see that the utility is currently in the SORTBLD phase and that the count of the number of rows shows the utility's progress. It also shows the number of objects listed and which object in this list is currently executing. You also see that the COPY, SORT, and RUNSTATS phases have completed and that we are waiting for the index to be rebuilt.

This display provides valuable information and allows you to determine the details of the utility's execution and helps you to determine if the utility is close to completion.

*Example 14-6 DISPLAY UTILITY output*

---

```
DSNU105I  -DB9A DSNUGDIS - USERID = DB2R8
          MEMBER =
          UTILID = MARY00
          PROCESSING UTILITY STATEMENT 1
          UTILITY = REORG
          PHASE = SORTBLD   COUNT = 2497592
          NUMBER OF OBJECTS IN LIST = 1
          LAST OBJECT STARTED = 1
          STATUS = ACTIVE
DSNU347I  -DB9A DSNUGDIS -
          DEADLINE = NONE
DSNU384I  -DB9A DSNUGDIS -
          MAXRO = 20 SECONDS
          LONGLOG = CONTINUE
          DELAY = 1200 SECONDS
DSNU111I  -DB9A DSNUGDIS - SUBPHASE = COPY COUNT = 54402
DSNU111I  -DB9A DSNUGDIS - SUBPHASE = SORTOUT COUNT = 1365504
DSNU111I  -DB9A DSNUGDIS - SUBPHASE = RUNSTATS COUNT = 13890
DSNU111I  -DB9A DSNUGDIS - SUBPHASE = RUNSTATS COUNT = 1108
DSNU111I  -DB9A DSNUGDIS - SUBPHASE = BUILD COUNT = 1363740
DSNU111I  -DB9A DSNUGDIS - SUBPHASE = SORTOUT COUNT = 1415680
DSNU111I  -DB9A DSNUGDIS - SUBPHASE = BUILD COUNT = 1409563
```

---

There are also some recent enhancements that have been made to the messages in the job output that have the Julian date and timestamp when the message is issued. Example 14-7 shows a partial output from a REORG utility using the SORTNUM elimination feature. Notice the Julian date and timestamp in the sample output.

*Example 14-7 SYSPRINT messages depicting Julian date and timestamp*

---

```

DSNU3340I  293 15:20:54.18 DSNURPIB - UTILITY PERFORMS DYNAMIC ALLOCATION OF SORT DISK SPACE
DSNU3342I  293 15:20:54.60 DSNURPIB - NUMBER OF OPTIMAL SORT TASKS = 5, NUMBER OF ACTIVE SORT TASKS = 5
DSNU395I   293 15:20:54.60 DSNURPIB - INDEXES WILL BE BUILT IN PARALLEL, NUMBER OF TASKS = 15
DSNU400I   293 15:21:08.07 DSNURPID - COPY PROCESSED FOR TABLESPACE MTSTBP.MTSTBP
                NUMBER OF PAGES=67931
                AVERAGE PERCENT FREE SPACE PER PAGE =  4.80
                PERCENT OF CHANGED PAGES =100.00
                ELAPSED TIME=00:00:17

```

---

In addition, the -DISPLAY UTILITY command has been enhanced to show the progress of LOGAPPLY while a RECOVER utility is running.

## Claims and drains

Claims and drains are mechanisms that DB2 uses to control concurrency. DB2 lays a claim on an object as an indication that the object is being accessed. DB2 takes a drain on an object to prevent any new claims on that object, but it allows existing claims to continue and be released when they have completed accessing that object.

## UTSERIAL lock

DB2 obtains a UTSERIAL lock to serialize all utilities running in the system. Only one UTSERIAL lock resource exists and it is acquired by DB2 in different situations, that is, when a utility enters the UTINIT or UTILTERM phase and when a utility updates SYSUTILX.

During the UTSERIAL lock's duration, DB2 performs different functions:

- ▶ Checks the compatibility of the utility with other work running in the system.
- ▶ Validates different DBET objects.
- ▶ Updates utility progress in SYSUTILX.
- ▶ Commits any updates.
- ▶ Releases the lock.

When you run too many utilities concurrently, you run the risk of multiple utilities contending for the same lock. As a result, a utility may time out and abend with a RC00E40085. This error indicates that a utility serialization LOCK or UNLOCK request has received an error from IRLM.

The problem is evident in systems that run with a low service profile class associated with the batch job (dispatching priority) or in systems that are CPU constrained. With either of these conditions, a utility job may hold onto the UTSERIAL lock for longer periods of time, increasing the likelihood of running into utility contention with other utility workload running in the system.

If you experience RC00E40085 abends frequently, review the following:

- ▶ CPU usage on the system.
- ▶ WLM service class for utility batch jobs.
- ▶ IRLM service class (should be higher than all DB2 address spaces).
- ▶ DSNZPARM UTIMOUT to allow more time for the lock to be acquired.

- ▶ APAR PK83996 (SPACE NOT BEING REUSED PROPERLY IN CATALOG AND DIRECTORY) may be applicable if there is a significant amount of space increase in SYSUTILX or SYSLGRNX prior to applying this APAR. One way to determine this is to check the output of image copies over several weeks to see if the size of these objects increased substantially.

### 14.1.3 Compatibility of utilities in restrictive states

For details about which utilities are compatible, refer to each utility's description in *DB2 Version 9.1 for z/OS Utility Guide and Reference*, SC18-9855. There is also a general topic on “Compatibility of utilities” in the *DB2 Version 9.1 for z/OS Monitoring and tuning DB2 performance*, SC18-9851.

### 14.1.4 Stopped utilities

When you issue DIS UTIL(\*) and view it in the stopped status, you only see the UTILID and not the jobname. All the information about the utilities that are currently running is kept in the directory table space SYSUTILX, but you cannot use SQL to access it.

However, you can run the DSNUTILB program with a DIAGNOSE DISPLAY SYSUTIL, which will dump the contents of SYSUTILX. The field USUJOBNM contains the JOBNAME.

### 14.1.5 Restarting utilities

A new DSNZPARM parameter, UTLRSTRT - UTILITY IMPLICIT RESTART, allows you to restart a utility job without having to specify the RESTART parameter within the JCL.

This feature allows DB2 to check for the UTILID in SYSUTILX. If the UTILID does not exist in SYSUTILX, then the utility starts from the beginning; otherwise, it starts from the beginning of the phase where the failure occurred, assuming you did not update the job and add a RESTART parameter. The RESTART parameter takes precedence if one is coded.

For more information, refer to Table 14-8 on page 402.

## 14.2 Parallelism

An important consideration in reducing elapsed time for a utility is maximizing the degree of parallelism, if possible. DB2 allows parallelism for many utilities, including, but not limited to, the LOAD, REORG, and REBUILD utilities.

### 14.2.1 Factors that determine the degree of parallelism

DB2 uses different criteria to determine the optimal degree of parallelism based on a number of factors:

- ▶ The number of page sets
- ▶ Total virtual storage available
- ▶ The number of processors (CPUs)
- ▶ The number of DB2 threads
- ▶ Buffer pool sizes and thresholds

LOAD, REORG, and REBUILD limit the degree of parallelism at three times the number of CPUs on the LPAR. UNLOAD limits the degree of parallelism to one times the number of CPUs on the LPAR.

Parallel Index Build by the LOAD/REORG/REBUILD utilities is not limited by the number of CPUs, as the DFSORT memory requirements generally limit parallelism.

**Note:** The number of CPUs does not include the number of zIIP or zAAP processors.

The following subtasks/phases can exploit parallelism:

- ▶ UNLOAD
- ▶ RELOAD
- ▶ SORT
- ▶ MERGE
- ▶ BUILD

Some of these tasks may be combined, for example, SORTBLD. Often, two tasks are assigned per index, one for UNLOAD and one for SORTBLD.

If inline statistics are requested, then additional tasks will be attached.

**Note:** PK26989 introduced a change where utility subtasks are *not* counted against either the IDBACK or the CTHREAD thresholds.

## 14.2.2 Determining if parallelism is constrained

The utility job output provides information about the degree of parallelism and an indication of whether parallelism was constrained. Look for any of the following messages as an indication of the degree of parallelism and if the utility job is at all constrained:

- ▶ DSNU364I - *csect-name* PARTITIONS WILL BE LOADED IN PARALLEL, NUMBER OF TASKS = nnnn

The number of parallel tasks is defined by nnnn.

- ▶ DSNU395I - *csect-name* - INDEXES WILL BE BUILT IN PARALLEL, NUMBER OF TASKS = nnnn

The number of utility tasks is defined by nnnn.

- ▶ DSNU397I - *csect-name* - NUMBER OF TASKS CONSTRAINED BY yyyy

Message DSNU397I in particular provides information about the reason why parallelism is constrained. If the reason is related to virtual storage, try to increase the region size for the job. Refer to 14.1.1, “Preparing for utility execution” on page 376 for more information.

The following messages indicate the specific reason for the constraint, where yyyy is CPUS, VIRTUAL STORAGE, or DD STATEMENTS:

- Message DSNU397I - *csect-name* - NUMBER OF TASKS CONSTRAINED BY CPUS

This error indicates the degree of parallelism is reduced by number of available processors. If another member is available with additional CPUs, run the utility on that member.



- Message DSNU397I - *csect-name* - NUMBER OF TASKS CONSTRAINED BY VIRTUAL STORAGE

This error indicates that the degree of parallelism is reduced by limited storage. You might consider changing REGION to 0M to use all available storage or increase to a larger REGION size.

- Message DSNU397I - *csect-name* - NUMBER OF TASKS CONSTRAINED BY VIRTUAL STORAGE BELOW

This error indicates the degree of parallelism is reduced by limited storage below the 16 MB line. Review your JCL to see if you can reduce the number of DD statements. If you are using sort work, consider using the SORTNUM elimination process, which is documented in Chapter 6, “Sort processing” on page 141. You might also try increasing the REGION size, especially if you specified a value less than 16 MB.

- Message DSNU397I - *csect-name* - NUMBER OF TASKS CONSTRAINED BY DD STATEMENTS

This error indicates the sort work files were allocated by JCL and there were fewer workfile DDs than the number of parallel tasks. Use the SORTNUM elimination and dynamic allocation for sort work data sets.

- DSNU397I - *csect-name* - NUMBER OF TASKS CONSTRAINED BY CONNECTION

This error indicates that additional tasks could have been used, but there are not enough free threads available to DB2. You should check DSNZPARMs IDBACK and CTHREAD and determine if they could be increased.

- ▶ DSNU427I *csect-name* - OBJECTS WILL BE PROCESSED IN PARALLEL,  
NUMBER OF OBJECTS = nnnn

The number of objects used is indicated by nnnn.

For more information about parallelism, refer to Chapter 2, “Managing partitions” on page 27.

## 14.3 Concurrency

What is the optimal number and type of utility jobs that can be run concurrently? The answer is not straightforward and depends on a number of factors. Here we examine how you determine the different types of utilities you can run and then the number of utilities you can run concurrently.

### 14.3.1 Running utilities concurrently

There are some online utilities that allow other utilities to run concurrently along with applications issuing SQL statements that can also run concurrently on the same set of objects. This is all controlled by DB2 by using the following documented techniques:

- ▶ Review the compatibility and concurrency topic for each online utility.
  - For every object in the utility job, the topic outlines the claim classes that the utility must claim or drain.
  - There is a restrictive state that DB2 sets on the target object based on the specific utility.
  - This section also discusses the compatibility of the utility with the object.
- ▶ If two actions are compatible on a target object, they can run simultaneously on that object in separate applications.
- ▶ Compatibility sometimes depends on specific options of that utility.

Refer to *DB2 Version 9.1 for z/OS Utility Guide and Reference*, SC18-9855 for additional information about this topic.

## 14.4 Number of concurrent utility jobs

What is the optimal number of utility jobs that can run concurrently?

One consideration is the type of utility. If you are running multiple RECOVER utility jobs concurrently, each RECOVER job has to read the log. FLA is used for RECOVER, and because the FLA buffers are divided for each utility, up to a limit of 10, then running more than 10 RECOVER jobs concurrently will undoubtedly ensure that some run without the FLA buffers, and will therefore slow down the recovery process.

Some general guidelines are as follows:

- ▶ Run no more than 10 concurrent utility jobs in a subsystem.
- ▶ If you run into UTSERIAL lock timeouts (RC00E40085), try to identify the cause.
- ▶ You can increase the DSNZPARM UTIMOUT utility timeout multiplier to a higher value.
- ▶ Stagger the utility jobs rather than submitting them all at once.

## 14.5 Data sharing considerations

In a DB2 data sharing environment, there are a few guidelines to follow in order to optimize concurrency and performance. Here we discuss the following topics specific to a data sharing environment:

- ▶ Submitting utility jobs
- ▶ Terminating and restarting utility jobs
- ▶ Concurrency
- ▶ Utility performance in a DSG
- ▶ Mixed releases in a DSG

### 14.5.1 Submitting utility jobs

When you submit a utility job in a data sharing environment, you can specify the member name or the group attach name in the JCL. If you use the member name, then the job must be submitted on the system where the member resides. You can also use the group attach name if you do not care where that job runs.

To ensure the job runs on a specific system, you can use the appropriate JES2 or JES3 control statements in your JCL to route the job to a specific system.

### 14.5.2 Terminating and restarting utility jobs

There are special considerations for terminating and restarting utilities in a data sharing environment. Each utility running in a DSG must have a unique utility ID (UID).

## **TERM UTILITY**

You can only terminate a utility running in a member of a DSG on that member where it was started. If a DB2 subsystem fails while a utility is in progress, you must restart that DB2 subsystem, and then you can terminate the utility from any system.

## **Restarting a utility**

The same UID must be used to restart the utility. If DB2 fails, you must restart DB2 on either the same or another z/OS system before you restart the utility. However, in co-existence mode, you must restart a utility only on a member that is running the same DB2 release level as the member on which the utility job was originally submitted.

### **14.5.3 Concurrency**

In order to optimize the performance of your utility jobs running in a data sharing environment, review the following guidelines:

- ▶ Spread DB2 utility jobs for different objects across the DSG.
- ▶ If possible, keep concurrent and compatible DB2 utilities for the same object running in the same member to minimize lock contention and GBP access.
- ▶ Potential UTSERIAL lock timeout can affect concurrent DB2 utility jobs.

### **14.5.4 Utility performance in a DSG**

Data sharing members can affect the performance of a utility depending on the utility. For example, if you run an online REORG on one member of a DSG, the other members logs are read during the log apply stage. RECOVER may also try to read the other members logs.

If you are running many utility jobs at the same time in a DSG, you can potentially see an unusually long UTILTERM phase elapsed time for many jobs because of the number of partitions and data sets, updates to RTS and DB2 Catalog statistics, and possible DB2 catalog contention from other workload running in the DSG. In addition, there is a potential for DASD and sort work data sets contention. During UTILTERM, DBET updates need to be communicated to other members of the group and logged by each member.

If you need to run many RECOVER jobs, remember that there is a limit of 10 jobs running concurrently that can utilize FLA. If you need to run more than 10 RECOVER jobs at a time, it is better to spread these jobs across the DSG to make optimal use of FLA processing.

### **14.5.5 Mixed releases in a DSG**

When you run mixed releases in a DSG, it is typically referred to as coexistence mode. There are a few considerations to be aware of when running utility jobs in this mode.

## DSNUTILB utilities batch program

The utilities batch program DSNUTILB contains a release independent load module, also called DSNUTILB, as well as multiple release-dependent modules, named with the version number as part of the name. For example, with DB2 V8 or 9, these release dependent modules are named DSNUT810 and DSNUT910, respectively. In addition, there are multiple dependent utility load modules; refer to Table 14-5 for a summary of these load modules based on a DB2 9 installation.

*Table 14-5 DB2 9 utility features and load module names*

Feature	Load module name
CATMAINT	DSNU91LA
CHECK	DSNU91LB
COPY	DSNU91LC
DIAGNOSE	DSNU91LD
LISTDEF	DSNU91LE
LOAD	DSNU91LF
MERGECOPY	DSNU91LG
MODIFY	DSNU91LH
OPTIONS	DSNU91LI
QUIESCE	DSNU91LJ
REBUILD	DSNU91LK
RECOVER	DSNU971LL
REORG	DSNU91LM
REPAIR	DSNU91LN
REPORT	DSNU91LO
RUNSTATS	DSNU91LP
STOSPACE	DSNU91LQ
TEMPLATE	DSNU91LR
UNLOAD	DSNU91LS
COPYTOCOPY	DSNU91LT
EXEC SQL	DSNU91LU
B/R SYSTEM	DSNU91LV

In coexistence in a mixed-release data sharing environment, where you have DB2 V8 and DB2 9, you need to have both release-dependent modules available, DSNUT810 and DSNUT910. In addition, you must have all the utility-dependent load modules and their aliases for the utilities in order to operate across the DSG. This can be done in two different ways and an example of each is provided below:

- ▶ Changing STEPLIB in DSNUPROC
- ▶ Cross-copy into load libraries

## Changing STEPLIB in DSNUPROC

Changing the STEPLIB in DSNUPROC is a simple way to ensure that the older release-dependent load module is available. You merely have to add the library containing the release-dependent module to the STEPLIB concatenation.

A simple change needs to be made to DSNUPROC to add the DB2 V8 load library. Refer to Example 14-8.

*Example 14-8 Modification to DSNUPROC for release coexistence*

---

```
//DSNUPROC PROC LIB='DB9J9.SDSNLOAD',
//          SYSTEM=DB9J,
//          SIZE=0K,UID='',UTPROC=''
//*
//*****
//*
//* PROCEDURE-NAME:      DSNUPROC
//*
//* DESCRIPTIVE-NAME:    UTILITY PROCEDURE
//*
//* FUNCTION:  THIS PROCEDURE INVOKES THE ADMF UTILITIES IN THE
//*            BATCH ENVIRONMENT
//*
//* PROCEDURE-OWNER:     UTILITY COMPONENT
//*
//* COMPONENT-INVOKED:   ADMF UTILITIES (ENTRY POINT DSNUTILB).
//*
//* ENVIRONMENT:        BATCH
//*
//* INPUT:
//*   PARAMETERS:
//*       LIB      = THE DATA SET NAME OF THE DB2  PROGRAM LIBRARY.
//*                  THE DEFAULT LIBRARY NAME IS PREFIX.SDSNLOAD,
//*                  WITH PREFIX SET DURING INSTALLATION.
//*       SIZE     = THE REGION SIZE OF THE UTILITIES EXECUTION AREA.
//*                  THE DEFAULT REGION SIZE IS 2048K.
//*       SYSTEM   = THE SUBSYSTEM NAME USED TO IDENTIFY THIS JOB
//*                  TO DB2 .  THE DEFAULT IS "DB9J".
//*       UID      = THE IDENTIFIER WHICH WILL DEFINE THIS UTILITY
//*                  JOB TO DB2.  IF THE PARAMETER IS DEFAULTED OR
//*                  SET TO A NULL STRING, THE UTILITY FUNCTION WILL
//*                  USE ITS DEFAULT, USERID.JOBNAME.  EACH UTILITY
//*                  WHICH HAS STARTED AND IS NOT YET TERMINATED
//*                  (MAY NOT BE RUNNING) MUST HAVE A UNIQUE UID.
//*       UTPROC   = AN OPTIONAL INDICATOR USED TO DETERMINE WHETHER
//*                  THE USER WISHES TO INITIALLY START THE REQUESTED
//*                  UTILITY OR TO RESTART A PREVIOUS EXECUTION OF
//*                  THE UTILITY.  IF OMITTED, THE UTILITY WILL
//*                  BE INITIALLY STARTED.  OTHERWISE, THE UTILITY
//*                  WILL BE RESTARTED BY ENTERING THE FOLLOWING
//*                  VALUES:
//*           RESTART(PHASE) = RESTART THE UTILITY AT THE
//*                           BEGINNING OF THE PHASE EXECUTED
//*                           LAST.
//*           RESTART = RESTART THE UTILITY AT THE LAST
//*                     OR CURRENT COMMIT POINT.
```

```

/*
/* OUTPUT: NONE.
/*
/* EXTERNAL-REFERENCES: NONE.
/*
/* CHANGE-ACTIVITY:
/*
/*
/******
/*
/*DSNUPROC EXEC PGM=DSNUTILB,REGION=&SIZE,
/*          PARM='&SYSTEM,&UID,&UTPROC'
/*STEPLIB DD DSN=&LIB,DISP=SHR
/*          DD DSN=DSN810.SDSNLOAD,DISP=SHR <----- coexistence
/*
/******
/*
/* THE FOLLOWING DEFINE THE UTILITIES' PRINT DATA SETS
/*
/******
/*
//SYSPRINT DD SYSOUT=*
//UTPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
/*DSNUPROC PEND          REMOVE * FOR USE AS INSTREAM PROCEDURE

```

---

### ***Cross-copy into load libraries***

Another approach is to cross-copy the release-dependent modules into the load libraries of the other release. For example, copy DSNUT810 into the Version 9 load libraries, and copy DSNUT910 and all applicable utility-dependent load modules into the DB2 V8 load libraries.

The issue here is when you apply maintenance to these modules: You must redo the cross-copy every time you apply maintenance to these modules. Because this is error-prone, we recommend using the first approach rather than the second.

Example 14-9 illustrates a sample JCL procedure to perform the cross-copy.

*Example 14-9 Cross-copy JCL procedure*

---

```
CROSCOPY PROC D910TPRE='DSN910',
// D810TPRE='DSN810',
// RGN=4096K,SOUT='*'
//* *****
/* FOR EXECUTION OF IEBCOPY - DB2 POST-INSTALLATION ***
/* *****
//COPY EXEC PGM=IEBCOPY,REGION=&RGN
//SYSUT3 DD UNIT=SYSDA,SPACE=(CYL,(5,1))
//SYSUT4 DD UNIT=SYSDA,SPACE=(CYL,(5,1))
/* ***** DB2 TARGET LIBRARIES *****
/*
//D810LOAD DD DSN=&D810TPRE..SDSNLOAD,DISP=OLD
//D910LOAD DD DSN=&D910TPRE..SDSNLOAD,DISP=OLD
/*
// PEND
//COEXIST EXEC PROC=CROSCOPY
//SYSIN DD *
COPYMOD INDD=((D910LOAD,R)),OUTDD=D810LOAD
SELECT MEMBER=(DSNUT910) <-- add any utility-dependent modules here
COPYMOD INDD=((D810LOAD,R)),OUTDD=D910LOAD
SELECT MEMBER=(DSNUT810)
```

---

### New utility functions

If all the members of a DSG are not running at the same release, then you should avoid taking advantage of any new utility functions for that release. When running in coexistence mode, with DB2 Vn and Vn-1, as long as your utility options that are supported in DB2 Vn-1, utility job can attach to a DB2 Vn or DB2 Vn-1 member without issue, then you should not encounter any complications.

## 14.6 Buffer pool considerations

DB2 utilities run under DB2 control and run under the auspices of the buffer manager component and use the DB2 buffer pools. Further performance gains may be possible through BP tuning. This section addresses areas that may be of concern when running utilities.

- ▶ Page fix option
- ▶ Buffer pool write operations

For more information about the topic of buffer pools, refer to *DB2 9 for z/OS: Buffer Pool Monitoring and Tuning*, REDP-4604 with respect to buffer pool hit ratios and the degree of utility parallelism that can be achieved. In general, the larger the buffer pools the better, although care must be taken not to cause problems for the DB2 subsystem or the LPAR as a whole if the buffer pool is page fixed or if paging occurs.

A minimum of 128 pages of buffer pool storage is required for each parallel task. For example, a LOAD of a partitioned table space with a partitioning index and two NPIs where inline statistics and an inline image copy are generated would require  $128 \times 5 = 640$  pages in the buffer pool to achieve the maximum degree of parallelism.

In a data sharing environment, care must be taken to ensure the group buffer pools are sized correctly. Objects that have inter-DB2 R/W interest may cause pages to be first written to the Coupling Facility (CF) instead of DASD. Also, data sharing locks may need to be propagated to the CF. It is important to monitor the CF structures to make sure that the lock and GBP structures are well behaved.

## 14.6.1 Page fix option

You have the option to alter buffer pools to use the PAGE FIX (PGFIX) option, which permanently page fixes an entire buffer pool once it is allocated. You can save valuable CPU time with this option, as the operating system no longer has to fix the page in memory during I/O operations. This option is available as part of the ALTER BUFFERPOOL command.

**Note:** When page fixing a buffer pool, ensure you have enough real memory for it. Otherwise, you might introduce z/OS paging, which is expensive and outweighs the benefit of page fixing.

## 14.6.2 Buffer pool write operations

Table 14-6 contains the number of writes per single I/O operation during a utility execution. The number of writes depends on the size of the buffer pool and its page size. These numbers have increased significantly, allowing for more I/O per write operation.

*Table 14-6 Writes per single I/O operation during utility execution*

Page size	Number of buffers	Number of pages
4 K	BP>=80,000 pages	128
	BP<80,000 pages	64
8 K	BP>=40,000 pages	64
	BP<40,000 pages	32
16 K	BP>=20,000 pages	32
	BP<20,000 pages	16
32 K	BP>=10,000 pages	16
	BP<10,000 pages	8

## 14.7 Specific utility considerations

In this section, we discuss operational specifics for each utility, including SORT, REORG, LOAD, UNLOAD, COPY, RECOVER, and REPAIR.

### 14.7.1 SORT considerations

The SORTNUM parameter basically predetermines the number of sort work data sets DB2 should choose for a sort process within a utility where a sort phase exists. When DB2 decides on the degree of parallelism, it uses this number as one input to the decision. If you choose too low a value for SORTNUM, that utility is going to fail. If you choose too high a number, it



may not be an optimal choice. And from a usability standpoint, what works today may not work tomorrow.

DB2 provides the ability where you can avoid setting the SORTNUM parameter. The algorithm uses RTS, so ensure RTS is enabled before you test this setup in DB2 V8, or DB2 will do more calculations to compute the correct number of sort work data sets. RTS is automatically included as part of the DB2 catalog in DB2 9. See Chapter 6, “Sort processing” on page 141 for additional information about the SORT process with DB2 utilities.

## 14.7.2 REORG considerations

The best REORG is the one you do not have to do. There are tools in the marketplace that allow you to establish different criteria for conditional reorganization. Of course, IBM has the DB2 Automation Tool, which is especially useful for this purpose.

Here we discuss a few considerations of the REORG utility.

### When to run the REORG utility

If the underlying data is not accessed sequentially or uses primarily index-only access, you may actually consider *not* running REORG.

Here are a few reasons to run a REORG utility:

- ▶ If data is disorganized and access to data is mostly sequential.
- ▶ For space reclamation purposes and to compress data for new data values or added partitions (although this is less necessary now that you can use LOAD COPYDICTIONARY across partitions).
- ▶ To reset Online Schema changes.
- ▶ To remove pointer records for relocated rows.

Make better use of RTS information and catalog space statistics to determine when utilities should run. If you do not have a tool like DB2 Automation Tool to help determine when to run utilities based on predefined criteria, then consider the use of the DB2 supplied stored procedure - DSNACCOX to make intelligent decisions about running the utilities COPY, REORG, or RUNSTATS. Instead of using the stored procedure, you could also make use of the DB2 Administration Tool and its Performance Query function.

For more information about the use of DB2 Tools to generate utilities, refer to Appendix A, “DB2 Tools” on page 413.

### When is a REORG necessary

Often, a reorganization of an index may be necessary, so consider using REORG INDEX instead of REORG TABLESPACE. Online REORG does not directly affect availability, so elapsed time is generally not too much of a concern, but run these online REORGs during a period of low activity. If you are having issues with too many threads preventing the online REORG to complete, refer to “DB2 Utility Enhancement Tool for z/OS” on page 429. The DB2 Utility Enhancement Tool can help block incoming threads so that the online REORG can complete.

Another recommendation for a partitioned table space with a large number of partitions is to run a REORG on a subset of partitions rather than all of the partitions in a single REORG.

## Unavailable resource

An enhancement has been made to the REORG utility to automatically display claimers when REORG receives a resource unavailable, as shown in Example 14-10.

Example 14-10 *DISPLAY CLAIMERS* output

```
1DSNU000I   293 15:20:53.48 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = DB2R8.DB2R8X
DSNU1044I   293 15:20:53.55 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
ODSNU050I   293 15:20:53.56 DSNUGUTC - REORG TABLESPACE DB2R8DB.DB2R8TS1 REUSE LOG NO SORTDATA NOSYSREC SHRLEVEL
REFERENCE KEEPDICIONARY COPYDDN(SYSCOPY)
DSNU1122I -DB9A 293 15:21:53.58 DSNURFIT - JOB DB2R8X PERFORMING
REORG
      WITH UTILID DB2R8.DB2R8X UNABLE TO DRAIN DB2R8DB.DB2R8TS1.
      RETRY 1 OF 6 WILL BE ATTEMPTED IN 360 SECONDS
DSNU1122I -DB9A 293 15:28:53.76 DSNURFIT - JOB DB2R8X PERFORMING
.
.
.
REORG
      WITH UTILID DB2R8.DB2R8X UNABLE TO DRAIN DB2R8DB.DB2R8TS1.
      RETRY 6 OF 6 WILL BE ATTEMPTED IN 360 SECONDS
DSNU590I -DB9A 293 16:03:54.51 DSNURDRN - RESOURCE NOT AVAILABLE, REASON=X'00C200EA', ON DB2R8DB.DB2R8TS1 PROHIBITS
PROCESSING
DSNT360I -DB9A *****
DSNT361I -DB9A * DISPLAY DATABASE SUMMARY
          * GLOBAL CLAIMERS
DSNT360I -DB9A *****
DSNT362I -DB9A DATABASE = DB2R8DB STATUS = RW
          DBD LENGTH = 12104
DSNT397I -DB9A
NAME      TYPE PART  STATUS              CONNID  CORRID  CLAIMINFO
-----
DB2R8TS1 TS    0001 RW,UTRO              TSO     DB2R8    (WR,C)
-          AGENT TOKEN 490
DB2R8TS1 TS    0001 RW,UTRO              TSO     DB2R8    (CS,C)
-          AGENT TOKEN 490
.
.
.
DB2R8TS1 TS    0025 RW,UTRO              TSO     DB2R8    (CS,C)
-          AGENT TOKEN 490
DB2R8TS1 TS    0025 RW,UTRO              TSO     DB2R8    (WR,C)
-          AGENT TOKEN 490
DB2R8TS1 TS          RW,UTRO              TSO     DB2R8    (CS,C)
-          AGENT TOKEN 490
DB2R8TS1 TS          RW,UTRO              TSO     DB2R8    (WR,C)
-          AGENT TOKEN 490
***** DISPLAY OF DATABASE DB2R8DB ENDED *****
DSN9022I -DB9A DSNTDDIS 'DISPLAY DATABASE' NORMAL COMPLETION
DSNU012I   293 16:03:54.53 DSNUGBAC - UTILITY EXECUTION TERMINATED, HIGHEST RETURN CODE=8
```

## REORG non-consecutive partitions

The REORG table space syntax allows you to easily REORG a number of consecutive partitions. However, if you want to REORG a number of non-consecutive partitions, the syntax currently does not support it.

## REORG for partitioned table spaces

REORG is able to unload and reload partitions in parallel, thereby reducing the elapsed time for a REORG.

Refer to Chapter 8, “Reorganizing data” on page 205 for more information about the REORG utility.

### 14.7.3 LOAD considerations

A big usability enhancement to the LOAD utility is in APARs PK63324 and PK63325. This enhancement allows you to prime partitions with a compression dictionary. The new parameter for LOAD is COPYDICTIONARY, and it allows for you to insert a compression dictionary into new partitions. Before this enhancement existed, if the partitioning key was based on a date or timestamp, then when you inserted data into new partitions, and if you expected data compression, you had to LOAD or REORG to get compression.

COPYDICTIONARY allows you to copy the compression dictionary from an earlier partition into a new partition that you specify, in effect, priming a new partition with a compression dictionary from another partition. This allows you to bypass a REORG if all you need is to run it to create the compression dictionary.

This feature is only available with INTO PART REPLACE. Refer to Example 14-11, where we copy the compression dictionary from PART 5 into PARTs 6, 7, and 8.

*Example 14-11 LOAD COPYDICTIONARY*

---

```
LOAD RESUME NO COPYDICTIONARY 5
  INTO TABLE PART 6 REPLACE
  INTO TABLE PART 7 REPLACE
  INTO TABLE PART 8 REPLACE
```

---

Refer to APARs PK63324 and PK63325 for additional information.

### 14.7.4 UNLOAD utility

The UNLOAD utility, as well as different SQL statements, support skipping rows that may be locked by those transactions performing updates. This action should only be used when you can tolerate not having all of the table data unloaded, as it can result in inconsistencies.

This enhancement to the UNLOAD utility effectively allows you to support an SQL enhancement with isolation level CS, which skips rows that are locked for transactional updates. Be aware that if you use this feature, you may get fewer rows unloaded instead of the actual number of rows that qualified for your SQL statement. However, you do not have to wait for transaction locks.

The new parameter added to the UNLOAD utility is SKIP LOCKED DATA.

### 14.7.5 COPY considerations

There are a few operational considerations for the COPY utility:

- ▶ MRU management
- ▶ SCOPE PENDING

#### MRU management

The COPY utility buffer pool usage now uses MRU management of those pages read by the COPY utility to reduce the impact of using the COPY utility and the associated buffer pool. Normally, pages read into the buffer pool are paged out using a LRU mechanism, so pages can be stolen if they are not in use for some time. Typically, when pages are read into the buffer pool for the COPY utility, they are only read in order to copy them; it is highly unlikely that these pages are re-referenced again by a transaction. You do not want these pages to

take any precedence over pages read into a buffer pool by an application. Pages read into the buffer pool by an application have more of a chance to be re-referenced.

This is the main reason why DB2 changed its algorithm for the COPY utility process from LRU to MRU. These pages that are read in by the COPY utility will be stolen before any pages that are read in on behalf of transactions. MRU management is automatically used for the pages read into the buffer pool on behalf of the COPY utility and prevents these pages from dominating the buffer pool. In addition, these pages are now more likely to be stolen and the buffer pool hit ratio may improve.

## SCOPE PENDING

The COPY utility now allows SCOPE PENDING to be used as an option to allow all objects that are in a COPY PENDING or informational COPY PENDING status to be copied. This is a helpful availability option that allows for image copy operations on those objects that are not recoverable.

### 14.7.6 RECOVER utility

The RECOVER utility has a few DB2 9 enhancements that help from an operational perspective. These are discussed in this section:

- ▶ Recover to any PIT with consistency
- ▶ Modify recovery simplification
- ▶ RECOVER RESTOREBEFORE enhancement
- ▶ Multiple page sets in the same RECOVER job

#### Recover to any PIT with consistency

The ability to recover to any point in time (PIT) with consistency significantly reduces and may even eliminate the need to run the QUIESCE utility which can be disruptive to applications (refer to 10.3, “Recovering to a point in time with consistency” on page 253).

#### MODIFY RECOVERY simplification

The MODIFY RECOVERY utility allows you to clean up information in SYSIBM.SYSCOPY and SYSIBM.SYSLGRNX. Prior to DB2 9, you could use only the DELETE option, and only beyond a certain age or a certain date. Now you have the ability to specify what you want to keep rather than what you want to delete, which really makes much more sense from a usability perspective. There is a RETAIN option that allows to keep the last, for example, 10 image copies.

Why keep more recovery information around that is older than two months, when you only have two months of archive log information stored in the BSDS? There is not much point in keeping SYSLGRNX information around from, for example, 3 months ago, if you cannot recover SYSLGRNX information anyway because the archive logs are not known to DB2. This is when you should use the LOGLIMIT parameter.

Another usability option is GDGLIMIT. If you are taking image copies using GDGs and you have a GDG cycle of 30, then there is no point in keeping SYSCOPY information around for more than 30 ICs because that image copy data set is gone.

It is possible to specify a combination of these parameters. Refer to Chapter 10, “Recovering data” on page 251 for additional information about the MODIFY RECOVERY enhancements.

**Tip:** Remember to run the REORG utility against SYSLGRNX regularly in order to speed up the MODIFY utility.

## RECOVER RESTOREBEFORE

The use of RECOVER RESTOREBEFORE solves the problem where you have been trying to perform a RECOVER on an object and its last image copy is corrupted. Prior to DB2 9, you might have had to rename the corrupted image copy so DB2 could not find it in the catalog and then it would automatically revert back to an earlier image copy. The syntax for RECOVER has been improved to allow you to specify that DB2 should automatically revert to an earlier image copy if it exists without having to manually rename the latest image copy.

This enhancement allows you to specify a point in the log and DB2 must then select an image copy for RECOVER before this point. In order to force DB2 to choose an image copy prior to the last one, you merely need to specify RESTOREBEFORE with an RBA or LRSN just prior to the image copy.

If you used a system level backup, and you now want to fall back to a traditional image copy (taken before the system level backup) as your recovery base, this parameter would be useful.

### Multiple page sets in the same RECOVER job

If you run a RECOVER utility with multiple page sets and one table is rarely or never updated and another has many updates since the last image copy, you should be aware that both objects are not available until the RECOVER utility completes. This means that the availability of a table that is rarely updated is affected.

**Tip:** Split read only page sets in highly updated page sets into separate jobs versus putting them together when using the RECOVER utility; this leads to higher availability.

### Multiple RECOVER jobs

Because the RECOVER utility uses FLA and there is a maximum of 10 jobs per subsystem that can use FLA concurrently, you should never run more than 10 RECOVER jobs simultaneously. If you try to run more than 10 jobs, then you may run out of FLA buffers and the recovery process will definitely take longer, because it will use a slower LOGAPPLY process.

Refer to 14.5.4, “Utility performance in a DSG” on page 389 for more information about FLA during RECOVER operations and *DB2 9 for z/OS: Backup and Recovery I/O Related Performance Considerations*, REDP-4452.

## 14.7.7 REPAIR utility

There are a few recent enhancements to the REPAIR utility noted here:

- ▶ REPAIR LOCATE SHRLEVEL CHANGE
- ▶ REPAIR VERSIONS

### REPAIR LOCATE SHRLEVEL CHANGE

You can now run the REPAIR LOCATE utility with SHRLEVEL CHANGE against an index, table space, and index space. This is not a major issue, as most users are not updating data on a regular basis, but rather a serviceability issue. In the event a page or row needs to be updated, the change can be accomplished online without having to stop the object.

## REPAIR VERSIONS

If you use DSN1COPY with the OBIDXLAT option to move table spaces from one system to another, then REPAIR VERSIONS allows you to update the versions in the catalog and directory for the specified page set.

However, there are some situations that you should be aware of when using DSN1COPY, as it is possible to get unpredictable results. For example, say you are copying a table space TS1 from one development subsystem (DEV1) to another (DEV2). Assume you have made four sets of changes to TS1 and have placed each of those changes (ALTERs) in DDL1-4 with four distinct UOWs.

In the target system, DEV2, you applied these same ALTERs, in DDL1-4, but with a different number of UOWs. The result is a different version number for TS1 on DEV1 and TS1 on DEV2.

Table 14-7 shows a sequence of events where, on DEV1 and DEV2, the same changes were made, in the same order, but with different UOWs. On DEV1, TS1 is version 4, but on DEV2, the same table space is version 3. The sequence of events is the same, the resultant DDL is the same, but REPAIR VERSIONS will not work on TS1 in DEV2.

Table 14-7 Scenario using DSN1COPY with different versions

Scenario	DEV1		DEV2	
	DDL	Version number	DDL	Version number
Case1	Applied DDL1 COMMIT	1	Applied DDL1 COMMIT	1
Case2	Applied DDL2 COMMIT	2	Applied DDL2 COMMIT	2
Case3	Applied DDL3 COMMIT	3	Applied DDL3 Applied DDL4 COMMIT	3
Case4	Applied DDL4 COMMIT	4		

For additional information about this topic, refer to APAR PK79144 and “Updating Version Information when moving objects to another subsystem” in *DB2 Version 9.1 for z/OS Utility Guide and Reference*, SC18-9855.

## 14.8 Miscellaneous considerations

There are a few miscellaneous considerations to take into account for your DB2 subsystem. In this section, we discuss the following topics:

- ▶ Logging considerations
- ▶ Invalidate cached dynamic statements

### 14.8.1 Logging considerations

It is generally expected that DB2 utilities update page sets and therefore write records to the DB2 log for data integrity. For this reason, it is essential that DB2 active log data sets reside on fast DASD and that dual logging is in effect. At commit, the log write buffer is flushed. Any

delay in writing the log records to DASD, or any contention on the log write latch, will impact utility performance.

To reduce the utility's processing impact, it is important to minimize the log writes. One simple way to do this is to specify the LOG NO parameter when appropriate.

Although not generally an issue for utilities, DB2 9 provides the best log read performance, as the number of active log read buffers was increased from 15 to 120. Performance for reading the archive logs was also improved by supporting compression and striping.

## 14.8.2 Invalidate cached dynamic statements

Statements that reside in the dynamic statement cache remain there indefinitely. If you run RUNSTATS against an object, the statements associated to that object are automatically invalidated. Normally, RUNSTATS is used to collect valuable statistics used by the DB2 optimizer and can be resource intensive. However, you may specify RUNSTATS table space-name UPDATE NONE REPORT NO to invalidate associated statements in the dynamic statement cache without collecting any statistics. The next time an application issues one of these statements, the statement has to be re-prepared and uses the current catalog statistics to determine the access path.

APAR PK47083 now invalidates cached dynamic statements upon the completion of certain utilities. Before APAR PK47083, the successful execution of certain utilities left all dynamic statements associated with the object in the dynamic statement cache.

For example, if you have an index that is temporarily in a REBUILD PENDING state, the DB2 optimizer will not choose that index for an access path. As a result, the access path chosen may be a table space scan. If that index is in REBUILD PENDING for a short time while a load or another utility is running, then once the REBUILD PENDING status is reset, ideally the DB2 optimizer should consider an access path that might use that index. This works fine in most situations, except when you are caching dynamic statements; the access path chosen might be a table space scan, and this sub-optimal access path will subsequently be used and re-used as this dynamic statement is cached and remains in the dynamic statement cache indefinitely.

We have the ability to invalidate the dynamic statement cache with the RUNSTATS utility today, but with this PTF, the process is automated; once REBUILD PENDING is reset at the completion of a LOAD, all dynamic statements that have a dependency on that table are reset. The next time a statement is issued by the application, DB2 completes a full prepare so that the DB2 optimizer is allowed to choose that index.

PK47083 ensures that cached dynamic statements get invalidated after successful completion of LOAD (LOAD REPLACE and LOAD PART REPLACE), REORG TABLESPACE, and REORG INDEX. Any restricted states that may have existed before the utility runs are reset.

For REORG INDEX and LOAD RESUME, any statements in the dynamic statement cache that depend on the associated table are invalidated if no indexes for the table have restricted states when the utility completes.

**Note:** Statements in the dynamic statement cache are invalidated for those commands that leave objects inaccessible upon completion, or for utilities that complete successfully.

## Conclusion

If a utility is cancelled, the associated statement in the dynamic statement cache may not be invalidated. In order to invalidate the statement, you should run RUNSTATS UPDATE NONE REPORT NO following the failed utility. This process ensures that the statement is invalidated.

## 14.9 DSNZPARMs that affect utility execution

There are a number of DSNZPARMs that affect utility execution. They are documented here for completeness and many are discussed in other sections. You should review the settings with the DB2 systems programmer and the DBAs to ensure that you make the best decision for your DB2 environment while keeping your application needs in mind.

Here we document the DSNZPARMS for the following macros:

- ▶ DSN6SPRM
- ▶ DSN6SYSP

### 14.9.1 DSN6SPRM

In this section, we list (in Table 14-8) the various DSNZPARMs related to utilities that are included in the DSN6SPRM macro.

Table 14-8 DSNZPARMS for macro DSN6SPRM

DSNZPARM and description	Values	Recommendation
<b>HONOR_KEEPDICIONARY</b> Specifies whether DB2 honors the LOAD and REORG parameter KEEPDICIONARY when tables are converted from basic row format to reordered row format.	The default value of HONOR_KEEPDICIONARY is NO. This means DB2 ignores the LOAD and REORG parameter KEEPDICIONARY when tables are converted from basic row format to reordered row format. If HONOR_KEEPDICIONARY is set to a value of YES, DB2 honors the LOAD and REORG parameter KEEPDICIONARY.	
<b>IGNSORTN</b> <b>IGNORE SORTNUM STAT</b> The value of the IGNORE SORTNUM STAT field is meaningful only when you enter YES in the UT SORT DATA SET ALLOCATION field.	The value of the IGNORE SORTNUM STAT field is meaningful only when you enter YES in the UTSORTAL UT SORT DATA SET ALLOCATION field. Specify YES to indicate that DB2 should ignore use of the SORTNUM clause in utility control statements.	Use YES so you do not have to change any existing utility control statements. However, you may need to change existing utility jobs that specify hardcoded sort work data sets.
<b>MAX_UTIL_PARTS</b> MAX_UTIL_PARTS specifies the number of partitions of a compressed table space that LOAD processes without limit. See APAR PK51853 for additional information.	The default value of MAX_UTIL_PARTS is 254. Valid values are in the range of 254 through 4096.	Set MAX_UTIL_PARTS if required, but watch storage used above the bar.



DSNZPARM and description	Values	Recommendation
<b>RESTORE_RECOVER_FROMDUMP RESTORE/RECOVER</b> Specifies whether the system-level backup for the RESTORE SYSTEM and the RECOVER utilities is from the disk copy of the system-level backup (NO), or from the dump on tape (YES).	RESTORE_RECOVER_FROMDUM P = NO indicates that the system-level backup for the RESTORE SYSTEM and the RECOVER utilities is from the disk copy of the system-level backup. RESTORE_RECOVER_FROMDUM P = YES indicates the system-level backup for the RESTORE SYSTEM and the RECOVER utilities is from the dump on tape. The default is NO.	
<b>RESTORE_TAPEUNITS MAXIMUM TAPE UNITS</b> Represents the maximum number of tape units or tape drives that the RESTORE SYSTEM utility can allocate when restoring a system-level backup that has been dumped to tape.	The acceptable values are NOLIMIT or 1 - 255. The default is NOLIMIT. NOLIMIT means that the RESTORE SYSTEM utility will allocate as many tape units as necessary to restore the system-level backup. You can override the setting for MAXIMUM TAPE UNITS by executing the RESTORE SYSTEM utility statement with the TAPEUNITS keyword.	
<b>SEQCACH SEQUENTIAL CACHE</b> Specifies whether to use the sequential mode to read cached data from a 3990 controller. Acceptable values are: ▶ BYPASS, SEQ. Default: BYPASS Update: Option 11 on panel DSNTIPB ▶ DSNZPxxx: ▶ DSN6SPRM SEQCACH. If you accept the default, BYPASS, DB2 prefetch bypasses the cache. If you specify SEQ, DB2 prefetch uses sequential access for read activity. Many sites gain a performance benefit by specifying SEQCACH. Recommendation: Specify SEQCACH if you have current disk devices with good cache sizes.	The acceptable values are BYPASS (default) and SEQ. If you accept the default BYPASS, DB2 prefetch bypasses the cache. If you specify SEQ, DB2 prefetch uses sequential access for read activity. Many sites gain a performance benefit by specifying SEQCACH.	Specify SEQCACH if you have current disk devices with good cache sizes, especially if the units are Enterprise Storage System (ESS) or RAMAC Virtual Array (RVA).
<b>SEQPRES UTILITY CACHE OPTION</b> Specifies whether DB2 utilities that do a scan of a non-partitioning index followed by an update of a subset of the pages in the index allow data to remain in cache longer when reading data.	The default value is NO. If you specify YES, these DB2 utility prefetch reads remain in cache longer, possibly improving performance of subsequent writes in the following cases for a table with very large non partitioned indexes: LOAD PART integer RESUME REORG TABLESPACE PART	The utility cache option is useful only with RAMAC disk attached to the 3990 Model 6. If you specify NO, DB2 utilities use the 3990 cache the same way as any other application, which is specified in the SEQUENTIAL CACHE option, SEQCACH.

DSNZPARM and description	Values	Recommendation
<b>STATCLUS</b> <b>STATISTICS CLUSTERING</b> Specifies the type of clustering statistics that are to be collected by the RUNSTATS utility.	The default is ENHANCED. STATCLUS=ENHANCED allows for enhanced clustering statistics, which may result in better SQL access paths when there are duplicate values of the clustering index or when the data is in reverse clustering order. These enhanced clustering statistics may cause many access paths to change. The STATCLUS=STANDARD option results in the same clustering statistics as DB2 V8.	We recommend using the new ENHANCED option to allow for better and improved access path selection.
<b>STATHIST</b> <b>STATISTICS HISTORY</b> Specifies which inserts and updates are recorded in catalog history tables.	The acceptable values are SPACE, NONE, ALL, and ACCESSPATH. The default is NONE. <ul style="list-style-type: none"> <li>▶ Use any of the following values:</li> <li>SPACE: All inserts and updates that DB2 makes to space-related catalog statistics are recorded.</li> <li>ACCESSPATH: All inserts and updates that DB2 makes to ACCESSPATH-related catalog statistics are recorded.</li> <li>ALL: All inserts and updates that DB2 makes in the catalog are recorded.</li> <li>NONE: None of the changes that DB2 makes in the catalog are not recorded.</li> </ul>	
<b>STATROLL</b> <b>STATISTICS ROLLUP</b> Specifies whether the RUNSTATS utility aggregates the partition-level statistics, even though some parts may not contain data.	The acceptable values are YES or NO. The default is NO. Specifying STATROLL=YES enables the aggregation of partition-level statistics and helps the optimizer to choose a better access path.	For DB2 subsystems that have large partitioned table spaces and indexes, we recommend that you specify YES for STATISTICS ROLLUP.
<b>REAL TIME STATS</b> Specifies the time, in minutes, that DB2 waits before attempting to write out page set statistics to the real-time statistics tables.	The acceptable values are 1 to 1440 minutes. The default is 30.	
<b>SYSTEM-LEVEL BACKUP</b> Specifies whether the RECOVER utility should use system-level backups as a recovery base (in addition to image copies and concurrent copies) for object-level recoveries.	The acceptable values are NO or YES. The default is NO. Specify NO for SYSTEM-LEVEL_BACKUP if you do not take system-level backups with the BACKUP SYSTEM utility. Specify YES if you take system-level backups with the BACKUP SYSTEM utility.	

DSNZPARM and description	Values	Recommendation
<b>UTIL_TEMP_STORCLAS</b> Specifies whether the CHECK utilities pass the storage class, supplied in the subsystem parameters, to DFSMS. See APAR PK41711 for additional information.	The default value for UTIL_TEMP_STORCLAS is spaces, which indicates that no STORCLAS is specified for shadow data sets that are allocated by the CHECK utilities.	Set the value of the UTIL_TEMP_STORCLAS to a valid storage class that is in the active SMS configuration. For performance, ensure that shadow data sets do not go into the same storage group.
<b>UTILS_DUMP_CLASS_NAME</b> <b>DUMP CLASS NAME</b> Specifies the name of the DFSMSshm dump class that will be used by the RESTORE SYSTEM utility to restore from a system-level backup that has been dumped to tape.	Acceptable values are a blank or a valid DFSMSshm dump class name not exceeding eight characters in length. The default is blank. UTILS_DUMP_CLASS_NAME is also the dump class that will be used by the RECOVER utility to restore objects from a system-level backup that has been dumped to tape. The setting is applicable only when you specify YES in field 2, RESTORE/RECOVER. You can override the setting for DUMP CLASS NAME by executing the RESTORE SYSTEM utility statement or the RECOVER utility statement with the DUMPCCLASS keyword.	
<b>UTLRSTRT</b> <b>UTILITY IMPLICIT RESTART</b> Allows you to restart a utility job without having to specify the RESTART parameter within the JCL, assuming you did not update the job and add a RESTART parameter. The RESTART parameter takes precedence if one is coded.	If UTLRSTRT=OFF, then to restart a utility, the user has to specify the RESTART keyword along with the value CURRENT or PHASE to indicate which phase is used for the restart. If UTLRSTRT=ON, then you merely resubmit the utility job without having to specify the RESTART parameter in the JCL.	Set the parameter UTLRSTRT to ON to enable implicit restart and simplify DB2's restart capability.
<b>UTIMOUT</b> <b>UTILITY TIMEOUT</b> Specifies the number of resource timeout values that a utility or utility command waits for a lock or for all claims on a resource of a particular claim class to be released.	The acceptable values are 1 to 254. The default is 6. For example, if you use the default value of 6, a utility can wait six times longer than an SQL application for a resource. This option allows utilities to wait longer than SQL applications to access a resource. The value of UTILITY TIMEOUT is used as the default value for the RETRY parameter of DB2 utilities, such as CHECK INDEX and online REBUILD INDEX.	

DSNZPARM and description	Values	Recommendation
<b>UTSORTAL</b> <b>UT SORT DATA SET ALLOCATION</b> Specifies YES to indicate that DB2 uses real-time statistics to determine the sort work data set sizes if real-time statistics data is available. If you specify YES and real-time statistics are not available, DB2 utilities that invoke a sort (CHECK, LOAD, REBUILD, REORG, and RUNSTATS) use a space prediction algorithm for dynamically-allocated sort work data sets.	The acceptable values are YES or NO. The default is NO. If you specify NO, you either need to specify the SORTNUM clause in the control statement when you run these utilities or allocate sort work data sets explicitly using JCL.	The recommendation is to use UTSORTAL=YES.
<b>VOLTDEVT</b> <b>TEMPORARY UNIT NAME</b> Specifies the device type or unit name for allocating temporary data sets.	The acceptable values are a valid device type or unit name. It can be blank if the USE SMS field is YES. The default is SYSDA. The value of TEMPORARY UNIT NAME is the direct access or disk unit name that is used for the precompiler, compiler, assembler, sort, linkage editor, and utility work files in the tailored jobs and CLISTs.	

## 14.9.2 DSN6SYSP

In this section, we list (in Table 14-9) the various DSNZPARMs that are related to utilities that are included in the DSN6SYSP macro.

Table 14-9 DSNZPARMS for macro DSN6SYSP

DSNZPARM and description	Values	Recommendation
<b>CTHREAD</b> <b>MAX USERS</b> Specifies the maximum number of allied threads (threads started at the local subsystem) that can be allocated concurrently.	Acceptable values are 1 - 2000. The default is 200. Count the following items as separate users: <ul style="list-style-type: none"><li>▶ Each TSO user (whether running a DSN command or a DB2 request from DB2 QMF).</li><li>▶ Each batch job (whether running a DSN command or a DB2 utility).</li><li>▶ Each IMS region that can access DB2.</li><li>▶ Each active CICS transaction that can access DB2.</li><li>▶ Each connection from users of CAF and RRSAP.</li></ul> The total number of threads accessing data that can be allocated concurrently is the sum of the MAX USERS value and the MAX REMOTE ACTIVE value.	The maximum allowable value for this sum is 2000. When the number of users who are attempting to access DB2 exceeds the number you specify, excess plan allocation requests are queued. In most situations, the amount of real and virtual storage determines the maximum number of threads that DB2 can handle. Note: The utility subtasks do not count against the thresholds, so application threads will not be impacted. However, IDBACK and CTHREAD and the current number of actual threads will be taken into account when deciding on the degree of parallelism for utilities. PK26989 introduced a change to not count utility subtasks against either the IDBACK or the CTHREAD thresholds. Consider increasing the values of subsystem parameters that control threads, such as MAX BATCH CONNECT and MAX USERS if your utility job is constrained by the number of connections.

DSNZPARM and description	Values	Recommendation
<b>DLDLFREQ</b> <b>LEVELID UPDATE FREQ</b> Controls how often, in the number of checkpoints, the level ID of a set or partition is updated.	Acceptable values are 0 to 32767. The default is 5. DLDLFREQ =0 disables down-level detection. The level ID frequency also controls how often the value that the RECOVER LOGONLY utility uses as the starting point for LOGAPPLY is updated. This value is updated at the same frequency (in checkpoints) as the level ID update frequency that you specify in this field.	Consider the following questions when you choose a value for the frequency of level ID updates: <ul style="list-style-type: none"> <li>▶ How often do you use backup and restore methods outside of DB2's control (such as DSN1COPY or DFDSS dump and restore)? If you rarely use such methods, you do not need to update the level ID frequently.</li> <li>▶ How many sets are open for update at the same time? If DB2 updates level IDs frequently, you have extra protection against down-level sets. However, if the level IDs for many sets must be set at every checkpoint, you might experience a performance degradation.</li> <li>▶ How often does the subsystem take checkpoints? If your DB2 subsystem takes frequent system checkpoints, you can set the level ID frequency to a higher number.</li> </ul>
<b>IDBACK</b> <b>MAX BATCH CONNECT</b> Specifies the maximum number of concurrent connections that are identified to DB2 from batch.	Acceptable values are 1 - 2000. The default is 50. Count each of the following items as a separate connection: <ul style="list-style-type: none"> <li>▶ Each batch job that uses DB2 QMF.</li> <li>▶ Each batch job that uses the DSN command processor.</li> <li>▶ Each task that is connected to DB2 through the call attach facility, which runs in batch.</li> <li>▶ Each RRSF connection that runs in batch.</li> </ul> Batch job requests to access DB2 that exceed this limit are rejected.	Note: The utility subtasks do not count against the thresholds, so application threads will not be impacted, but IDBACK and CTHREAD and the current number of actual threads will be taken into account when deciding on the degree of parallelism for utilities. PK26989 introduced a change to not count utility subtasks against either the IDBACK or the CTHREAD thresholds. Consider increasing the values of subsystem parameters that control threads, such as MAX BATCH CONNECT and MAX USERS if your utility job is constrained by the number of connections.
<b>LOGAPSTG</b> <b>LOG APPLY STORAGE</b> The value in this field represents the maximum DBM1 storage that can be used by the Fast Log Apply process.	The acceptable values are 0 to 100M. The default is 100M. If you specify 0, the Fast Log Apply process is disabled except during DB2 restart. During DB2 restart, the Fast Log Apply process is always enabled.	Specify 10 MB of LOGAPPLY storage for each concurrent RECOVER job that you want to have faster log apply processing. The default value of 100 MB provides log apply storage for 10 concurrent RECOVER jobs. We recommend that you use the default value of 100M.

DSNZPARM and description	Values	Recommendation
<p><b>MGEXTSZ</b>  <b>OPTIMIZE EXTENT SIZING</b>  Specifies whether secondary extent allocations for DB2-managed data sets are to be sized according to a sliding scale that optimizes the likelihood of reaching the maximum data set size before secondary extents are exhausted.</p>	<p>Acceptable values: are YES or NO. The default is YES.  For MGEXTSZ=YES, if you specify the default value of YES, DB2 automatically optimizes the secondary extent allocations of data sets for table spaces and index spaces that have a SECQTY value of greater than zero. When all secondary extents are exhausted for the first data set of a non-partitioned table space or a non-partitioned index space that has a SECQTY value of greater than zero, the primary space allocation of each subsequent data set is the larger of the SECQTY setting and the value that is derived from the sliding scale algorithm. If you select NO, you manage secondary extent allocations manually.</p>	<p>We recommend that you keep the default value of YES.</p>
<p><b>URLGWTH</b>  <b>UR LOG WRITE CHECK</b>  Specify the number of log records that are to be written by an uncommitted unit of recovery (UR) before DB2 issues a warning message to the console.</p>	<p>Acceptable values are 0 to 1000K. The default is 0.  The purpose of this option is to provide notification of a long-running UR. Long-running URs might result in a lengthy DB2 restart or a lengthy recovery situation for critical tables. Specify the value in 1-K (1000 log records) increments. A value of 0 indicates that no write check is to be performed.</p>	<p>We recommend specifying a non-zero value for this parameter, which can alert you to potential applications that may cause utilities like REORG SHRLEVEL CHANGE to fail.</p>







# Part 4

## Appendixes

This part includes the following reference documentation:

- ▶ Appendix A, “DB2 Tools” on page 413
- ▶ Appendix B, “Utilities related maintenance” on page 465





## DB2 Tools

There are a number of DB2 Tools that can assist you with utility generation, additional utility features, and some special functions related to DB2 catalog statistics. In this appendix, we discuss some of these features at a high level. We focus on the capabilities of each of the tools while keeping DB2 utilities as our primary focus.

A resource for detailed information about each tool can be found at the following address:

<http://www.ibm.com/software/data/db2imstools/>

The tools are:

- ▶ DB2 Administration Tool
- ▶ DB2 Utility Enhancement Tool for z/OS
- ▶ DB2 Automation Tool
- ▶ DB2 High Performance Unload for z/OS
- ▶ DB2 Recovery Expert for z/OS
- ▶ DB2 Change Accumulation Tool
- ▶ DB2 Storage Management Utility
- ▶ Optim Query Tuner and Optim Query Workload Tuner
- ▶ DB2 Buffer Pool Analyzer for z/OS

## A.1 DB2 Administration Tool

The DB2 Administration Tool allows you to perform specific functions in conjunction with all of the DB2 utilities. This tool serves many purposes: catalog navigation, migration of objects, DDL generation, change management, and so on. From a DB2 utility perspective, the following capabilities exist to help DBAs in their day to day interaction with DB2:

- ▶ Generate LISTDEFS and TEMPLATES for utility generation.
- ▶ Generate *ad hoc* DB2 utilities.
- ▶ Generate offline DB2 utilities.
- ▶ Migrate DB2 catalog statistics.
- ▶ DB2 Performance Queries.

For more information about the DB2 Administration Tool, refer to the following address:

<http://www.ibm.com/software/data/db2imstools/db2tools/db2admin/>

The Administration main menu ADB2 is provided in Example A-1.

### *Example A-1 Administration main menu*

---

```
ADB2 dmin ----- DB2 Administration Menu 7.2.0 ----- 20:
Option ==>
```

1 - DB2 system catalog	DB2 System: DB9A
2 - Execute SQL statements	DB2 SQL ID: DB2R8
3 - DB2 performance queries	Userid : DB2R8
4 - Change current SQL ID	DB2 Rel : 915
5 - Utility generation using LISTDEFS and TEMPLATES	
P - Change DB2 Admin parameters	
DD - Distributed DB2 systems	
E - Explain	
Z - DB2 system administration	
SM - Space management functions	
W - Manage work statement lists	
X - Exit DB2 Admin	

Interface to other DB2 products and offerings:

D - DB2I

---

### A.1.1 Generate LISTDEFS and TEMPLATES for utility generation

The DB2 Administration Tool has a convenient process to generate LISTDEFS and TEMPLATES for utility processing. These can be generated once and reused over and over by all admin users. This option is conveniently provided in the Administration main menu as option 5 (refer to Example A-1).

In Example A-2, we provide an example of a user-defined template for the UNLOAD data set used by REORG INDEX or REORG TABLESPACE. There are many options available to define these templates and, later on, when you invoke a specific utility, you have the option to specify that you prefer to use templates in the utility generation. Refer to Example A-4 on page 417 for more information about specifying template usage during *ad hoc* utility generation.

---

*Example A-2 Template example for UNLOAD data set*

---

```
ADB25TE n ----- DB9A Utility Template ----- 13:15
Command ==>
```

Enter name and optional remark:

```

TEMPLATE ==> CCUNLOAD (Template Name)
Remark    ==> Template for unload data set used by REORG INDEX and REORG TAB
LESPACE utility. Copy and rename this sample template to use it. Sample tem >
Common Options:
UNIT      ==> (Device Number, Type or Group Name)
DSN       ==> DB2.&SSID..&DB..&SN..D&JDAY.&HOUR.&MINUTE..UNLOAD

Change other common options ==> NO (Yes or No)
Change disk options         ==> NO (Yes or No)
Change tape options         ==> NO (Yes or No)

Statement ==> TEMPLATE CCUNLOAD DSN DB2.&SSID..&DB..&SN..D&JDAY.&HOUR.&MINU
TE..UNLOAD STACK N TRTCH NONE
```

---

## A.1.2 Generate ad hoc DB2 utilities

The DB2 Administration Tool allows you to generate different utilities at the database, table space, index, or table level. You can run the utility for either one or all objects in a list; this is achieved by using a utility line command or using the primary command UTIL.

In Example A-3, we illustrate this process by using a list of table spaces. In this screen, you can either enter UTIL as a primary command or UT as a line command; both give you another screen where you can specify the particular utility you want to run. If you know the utility in advance, then you could also use U.x, where x is the command for a particular utility. In this example, we use U.C for a full image copy utility. Your choices vary depending on the list of objects.

Example A-3 DB2 Table Spaces

```
ADB21S in ----- DB9A Table Spaces ----- Row 1 to 13 of 13
Command ==> util                               Scroll ==> PAGE

Commands: GRANT  MIG  DIS  STA  STO
Line commands:
  T - Tables  D - Database  A - Auth  G - Storage group  ICS - Image copy status
  DIS - Display table space  STA - Start table space  STO - Stop table space
  ? - Show all line commands
```

Select	Name	DB Name	Parts	Bpool	LESICTables	Act. pages	Segsz	T L
*	*	*	*	*	*	*	*	*
ut	MTSTBX	MTSTBX	25	BP2	S N A N Y	1	54355	0 Y
	MTSTBO	MTSTBO	25	BP2	S N A N Y	1	54355	0 Y
	MTSTBP	MTSTBP	0	BP2	S N A N Y	1	54329	4 Y
	SIDDAGO	SIDDAGOY	0	BP1	T N A N Y	1	-1	64 Y
	MAPTBS	SIDDAGOZ	0	BP0	T N A N Y	1	-1	64 Y
u.c	GLWSDPT	DB2R601	0	BP15	A N A N Y	1	134	32 Y
	GLWSEMP	DB2R601	4	BP15	A N A N Y	1	1081	0 Y
	GLWSEPA	DB2R601	0	BP15	A N A N Y	1	7236	64 Y
	GLWSPJA	DB2R601	0	BP15	A N A N Y	1	3002	32 Y
	GLWSPRJ	DB2R601	0	BP15	A N A N Y	1	1269	32 Y
	GLWSSPL	DB2R601	4	BP15	A N A N Y	1	144	0 Y
	GLWS001	DB2R601	0	BP15	A N A N Y	7	60	4 Y
	GLWS002	DB2R601	0	BP15	A N A N Y	4	372	4 Y

```
***** END OF DB2 DATA *****
```

If we used the UTIL primary command, the next screen allows you to choose the specific utility. See Example A-4 on page 417 for the utility specification screen, where the choice depends on the originating object type, which in our example is a table space. In addition, notice at the bottom of the screen that there are several options:

- ▶ Review/change options
  - YES: Allows you to bypass the specification of options for the specific utility. For example, once you set up options for a full image copy, you need not go to the COPY screen any longer by specifying YES.
  - NO: Allows you to review and possibly change the current settings.
- ▶ Generate work statement list
  - YES: Allows you to add the utility invocation to a work statement list (WSL). A WSL is useful if there are several functions you want to run together. Use the W option within the primary DB2 Administration Tool menu to access the WSL screen, as shown in Example A-1 on page 414.
  - NO: Allows you to generate the JCL for the utility immediately.

- ▶ Generate template statements
  - YES: Allows you to generate template statements during the utility execution.
  - NO: Does not allow you to generate template statements during the utility execution.
- ▶ Generate modify after copy
  - YES: Allows you to generate a MODIFY utility step after the COPY step.
  - NO: Just generate a COPY utility without including a MODIFY utility.

*Example A-4 Table Space Utilities screen*

---

ADB2US in ----- DB9A Table Space Utilities ----- 13:33  
 Option ==>

Execute utility on		DB2 System: DB9A
all the selected table spaces		DB2 SQL ID: DB2R8
		More: +
C - Copy full	CI - Copy incremental	C2 - Copytocopy
CC - Copy concurrent		
E - Mergecopy	EN - Mergecopy newcopy	
K - Check index	KD - Check data	KL - Check LOB
M - Modify		
N - Repair nocopypend	NA - Repair nocheckpend	NB - Repair norcvrpend
O - Reorg	OU - Reorg unload only	
OC - Reorg w/Inline Copy		
P - Report recovery	Q - Quiesce	
R - Runstats	RT - Runstats table all	RR - Runstats report
RX - Runstats (to invalidate dynamic cache)		
V - Recover		VG - Recover to last GDG
VI - Rebuild index	VR - Recover torba	VL - Recover logonly
DG - Define GDG for copy data sets		VP - Recover tologpoint
U - Unload		

Utility control options:

Review/change options	: Yes	(Yes/No)
Generate work statement list	: No	(Yes/No)
Generate template statements	: Yes	(Yes/No)
Generate modify after copy	: NO	(Yes/No)

---

Just to illustrate one example where the specific options for a given utility are displayed, we choose the Copy Full option and illustrate the different options for the COPY utility in Example A-5. There are more COPY options; here we just illustrate the first one.

*Example A-5 Copy utility options*

---

```
ADB2USC n ----- DB9A Specify Utility Options - COPY DATA FULL ----- 17:20
Option ==>
```

```
Execute utility on table space DB2R601.GLWSDPT
using the following options:
```

FULL	==> Y	(Type of image copy, Y-Full(default), N-Incremental)
CHANGELIMIT	==>	(Yes/No/Any, take image copy on specified limit)
pct-value1	==>	(First of value range, 0.0-100.0, default is 1)
pct-value2	==>	(Second of value range, 0.0-100.0, default is 10)
REPORTONLY	==>	(Yes/No, display image copy info only)
PARALLEL	==>	(Yes/No, process objects in parallel)
CHECKPAGE	==>	(Yes/No, check validity of each page in index space)
CONCURRENT	==>	(Yes/No, use DFSMS concurrent copy to make image)
SHRLEVEL	==>	(Type of access allowed other programs during execution, R-Reference(default), C-Change)
CLONE	==>	(Yes/No, copy clone table only)
SCOPE	==>	(Limit scope of objects to be copied,

---

### A.1.3 Generate offline DB2 utilities

The DB2 Administration Tool allows you to generate offline DB2 utilities. These utilities are:

- ▶ DSN1COPY
- ▶ DSN1PRNT
- ▶ DSN1COMP

To generate any of these offline utilities, you have to use the DSN1 command. To do so for a table space, enter the line command SP in the Table Space screen to get to the Table Space Partitions screen, as shown in Example A-6 on page 419.



### Example A-6 SP command

ADB21S in ----- DB9A Table Spaces ----- Row 5 from 110  
Command ==> Scroll ==> PAGE

Commands: GRANT MIG DIS STA STO

Line commands:

T - Tables D - Database A - Auth G - Storage group ICS - Image copy status  
DIS - Display table space STA - Start table space STO - Stop table space  
? - Show all line commands

Select	Name G*	DB Name RAVIK*	Parts * *	Bpool * *	L E S I C * * * * *	Tables *	Act. pages *	Segsz * * *	T L * * *
	GLWSEPA	RAVIK	0	BP15	R N A N Y	1	396	64	Y
	GLWSDPT	RAVIK	1	BP15	A N A N Y	1	12	4	G Y
SP	GLWSEMP	RAVIK	4	BP15	A N A N Y	1	720	64	R Y
	GLWSPJA	RAVIK	0	BP15	A N A N Y	1	36	32	Y
	GLWSPRJ	RAVIK	0	BP15	A N A N Y	1	36	4	Y
	GLWSSPL	RAVIK	4	BP15	A N A N Y	1	144	0	N
	GLWS001	RAVIK	0	BP15	S N A N Y	7	60	4	Y
	GLWS002	RAVIK	0	BP15	A N A N Y	4	372	4	Y
	GLWSEMP0	RAVIK	4	BP15	A N T N Y	0	-1	0	Y

\*\*\*\*\* END OF DB2 DATA \*\*\*\*\*

After pressing Enter, the Table Space Parts screen, ADB21SP, opens, as shown in Example A-7. From here, you can use the line command "UT" to get to the Utility screen.

### Example A-7 Table Space Parts screen

ADB21SP n -- DB9A Table Space Parts for RAVIK.GLWSE > ----- Row 1 to 4 of 4  
Command ==> Scroll ==> PAGE

Line commands:

T - Tables D - Database S - Table space G - Storage group A - Auth  
DIS - Display database STA - Start database STO - Stop database  
? - Show all line commands

Select	Part *	Prim Qty *	Sec Qty * *	T Group * * *	Storage VCAT *	Card *	Pct Act *	Pct Drop * *	F *
	1	180	180	I	GLWG01 DB9AU	0	0	0	R
UT	2	180	180	I	GLWG01 DB9AU	0	0	0	R
	3	180	180	I	GLWG01 DB9AU	0	0	0	R
	4	180	180	I	GLWG01 DB9AU	0	0	0	R

\*\*\*\*\* END OF DB2 DATA \*\*\*\*\*

In the Table Space Utilities screen, ABD2US, you can see the command DSN1 in the body of the panel. Refer to Example A-8, where we enter DSN1 as a primary command.

*Example A-8 Table Space Utilities screen: DSN1 option*

---

ADB2US in ----- DB9A Table Space Utilities ----- 11:52  
Option ==> DSN1

Execute utility on		DB2 System: DB9A
table space RAVIK.GLWSEMP part	2	DB2 SQL ID: DB2R8
		More: +

C - Copy full	CI - Copy incremental	C2 - Copytocopy
CC - Copy concurrent		
E - Mergecopy	EN - Mergecopy newcopy	
K - Check index	KD - Check data	KL - Check LOB
M - Modify		
N - Repair nocopypend	NA - Repair nocheckpend	NB - Repair norcvrpend
NL - Repair Levelid		
O - Reorg	OU - Reorg unload only	OO - Online reorg
OC - Reorg w/Inline Copy		
P - Report recovery	Q - Quiesce	
R - Runstats	RT - Runstats table all	RR - Runstats report
RX - Runstats (to invalidate dynamic cache)		
V - Recover	VC - Recover tocopy	VG - Recover to last GDG
VI - Rebuild index	VR - Recover torba	VL - Recover logonly
DG - Define GDG for copy data sets		VP - Recover tologpoint
U - Unload		

DSN1 - Offline utilities

Utility control options:

Review/change options	: YES	(Yes/No)
Generate work statement list	: NO	(Yes/No)
Generate template statements	: NO	(Yes/No)
Generate modify after copy	: YES	(Yes/No)

---

Finally, after pressing Enter, we get the Offline Utilities Selection screen, ADB2US1 as shown in Example A-9. In this screen, you can now choose between DSN1PRNT, DSN1COPY, and DSN1COMP, along with different parameter options. You also have the option to STOP or START the table space before or after execution.

*Example A-9 Offline Utilities Selection screen*

---

```
ADB2US1 n ----- DB9A Offline Utilities Selection ----- 11:56
Option ==>
```

```
Select an offline utility for execution on object
table space RAVIK.GLWSEMP part      2
```

```
1P - DSN1PRNT, to print the data set
```

```
1C - DSN1COPY, to copy the data set
```

```
For this selection, please provide SYSUT2 data set information:
```

```
Name      ==> DUMMY
```

```
Disp      ==> (NEW/OLD - ignored if DUMMY data set)
```

```
Space units ==> (Trk,Cyl)
```

```
Primary space ==> (Default 1)
```

```
Sec. space ==> (Default 1)
```

```
Unit type ==> (Default SYSDA)
```

```
1M - DSN1COMP, to estimate space savings using data compression
```

```
-STOP the table space before execution ==> NO (Yes/No)
```

```
-START the table space after execution ==> NO (Yes/No)
```

---

In case you want to run any of these utilities for an index, from the Indexes screen, use the XP line command to display the Index Partitions screen (ADB21XP). Proceed with line command UT and then primary command DSN1 as outlined in this section. The only utilities available for index spaces are DSN1PRNT and DSN1COPY; DSN1COMP is not available.

### A.1.4 Migrate DB2 catalog statistics

You can migrate catalog statistics to other systems using the MIG function in the DB2 Administration Tool. The MIG primary command has the option shown in Example A-10 to migrate catalog statistics as part of the Scope of migrate.

### Example A-10 Migrate catalog statistics

```
ADB28M in ----- DB9A Migrate Parameters ----- 18:49
Option ==>
```

```

Please specify the following for DB2 Admin Migrate:      DB2 System: DB9A
                                                         DB2 SQL ID: DB2R8
                                                         More:      +
Worklist name . . . . . : DB2CAT    (also used as middle qualifier in DSNs)

```

```
Data set information:
  PDS for batch jobs   . . . : MARY.CNTL
  Prefix for datasets  . . . : DB2R8
```

[illegible]

```

Migrate options:
Generate MIG jobs in batch . . . . : YES      (Yes/No)
Generate work stmt list . . . . . : NO       (Yes/No)
Use masking for batch migrate . . : YES      (Yes/No, N if stmt list is Y)
Combine job steps . . . . . : YES          (Yes/No)
    Member prefix for combined jobs : ADBMG   (default ADBMG )

Scope of migrate:
    DDL . . . . . : Y                      (Yes/No)
    Data . . . . . : N                    (Yes/No)
    Catalog statistics . . . . . : Y       (Yes/No)
DROP on target before CREATE . . : NO       (Yes/No, No if scp. is data only)
Create storage group . . . . . : NO        (Yes/No)
Generate GRANT statements . . . . : YES     (Yes/No)
Run SQLID . . . . . :                     (Blank, a SQLID, or <NONE>)
Unload method . . . . . : U               (U - Unload)

```

## A.1.5 DB2 Performance Queries

The performance queries that have to do with DB2 utilities in the DB2 Administration Tool are listed in Example A-11. We will not examine each option here, but a few of the more important ones are given some consideration and discussed here.

### *Example A-11 DB2 Performance Queries*

---

ADB23 min ----- DB2 Performance Queries ----- 19:21

Option ==>

- 1 - Table spaces without RUNSTATS within 0 days DB2 System: DB9A
  - 1X - Indexes without RUNSTATS within 0 days DB2 SQL ID: DB2R8
- RUNSTATS information is required for options 2 through 9
- 2 - Table spaces with more than 10 percent relocated rows
  - 3 - Indexes with clustering level problems
  - 4 - Table spaces with more than 5 percent dropped space
  - 5 - Table spaces with locking size = 'S' (table space locking)
  - 6 - Index with 2 or more levels
  - 7 - Indexes with 150 or more leaf page distance
  - 8 - Indexes on tables with fewer than 6 pages
  - 9 - Indexes not used by any plan or package
  - 10 - Table spaces containing more than one table
  - 11 - Table spaces without SPACE information
  - 11X - Indexes without SPACE information
- SPACE information is required for options 12 through 13
- 12 - Table spaces exceeding allocated primary quantity
  - 12X - Indexes exceeding allocated primary quantity
  - 13 - Allocated and used space for table spaces
- RTS Real-Time Statistics tables are required for options 14 and 14X.
- 14 - Table Space maintenance recommendations
  - 14X - Index Space maintenance recommendations

WHERE Database LIKE ==>

AND obj has more than ==> 32 pages

---

## Indexes with 150 or more leaf page distance

As one example, choose option 7, which locates those index objects with a leaf page distance of more than 150 pages. You can choose the number of pages. In this example, we chose 150 pages. In this list, we see 18 objects that fit this criteria, as shown in Example A-12. If you wanted to run a REORG for all of them, or for a few of them, you have both options available to do so. The DB2 Administration Tool will generate the utility JCL and you can then decide when to best schedule the job.

### Example A-12 Performance query option 7

```
ADB237 in---- DB9A Indexes with 150 or More Leaf Page Distan Row 1 to 15 of 18
Command ==> Scroll ==> PAGE
```

This panel shows indexes with 150 or more leaf page distance. The leaf page distance is defined as: 100 times the average number of pages between successive active leaf pages of the index. If this value exceeds 200, consider reorganizing the index. You might also consider redesigning the indexes. Things to consider are freespace/reorg frequencies and insert/update/delete patterns and frequencies.

Commands: 0 - Reorg UT - Utilities  
Line commands: S - Select 0 - Reorg

S Index Name	Index Owner	Part Table Name	Table Owner	Leaf Distance
*	*	* *	*	*
-----	-----	-----	-----	-----
XPLAN_TABLE1	PAOLOR3	0 PLAN_TABLE	PAOLOR3	210
GLWXDPT2	PAOLOR5	0 GLWTDPT	PAOLOR5	2530
GLWXDPT3	PAOLOR5	0 GLWTDPT	PAOLOR5	1520
GLWXDPT4	PAOLOR5	0 GLWTDPT	PAOLOR5	1075
GLWXEMP1	PAOLOR5	1 GLWTEMP	PAOLOR5	18588
GLWXEMP1	PAOLOR5	2 GLWTEMP	PAOLOR5	17898
GLWXEMP1	PAOLOR5	3 GLWTEMP	PAOLOR5	18471
GLWXEMP1	PAOLOR5	4 GLWTEMP	PAOLOR5	15762
GLWXEMP2	PAOLOR5	0 GLWTEMP	PAOLOR5	3251
GLWXEMP4	PAOLOR5	0 GLWTEMP	PAOLOR5	7338
GLWXEMP5	PAOLOR5	0 GLWTEMP	PAOLOR5	8854
GLWXEMP6	PAOLOR5	0 GLWTEMP	PAOLOR5	14706
GLWXPRJ2	PAOLOR5	0 GLWTPRJ	PAOLOR5	1911
GLWXPRJ3	PAOLOR5	0 GLWTPRJ	PAOLOR5	2811
RNDXEPA1	RAVI	0 GLWTEPA_RANDOM	RAVI	19915

## Performance queries using RTS

Options 14 and 14X are interesting, as they use RTS to generate the results.

### Performance query 14

If you choose Option 14, you get the screen “Table Space maintenance recommendations”, which is shown in Example A-13 on page 425. You can specify the criteria to be used or choose the defaults, which are provided in parenthesis.

*Example A-13 Performance query option 14: table space maintenance recommendations*

ADB2314T ----- DB9A Input Parameters for Real-Time Statistics ----- 19:54  
Option ==>

The input values specified below are used in the calculations which determine the recommended table space actions. For a full description of any parameter, use panel HELP and refer to the entry indicated by the parenthesized keyword.

Run using default settings: YES (Yes/No)	(default)	
	More:	+
Limit, number of physical extents . . . . . : (ExtentLimit)	(50)	
Limit, number of days since last image copy. . . . . : (CRDaySncLastCopy)	(7)	
Ratio, as percent, of updated pages to preformatted pages in table space or partition. . . . . : (CRUpdatedPagesPct)	(1)	
Ratio, as percent, of distinct updated pages to total active pages since last image copy . . . . . : (ICRUpdatedPagesPct)	(1)	
Ratio, as percent, of INSERTs, UPDATEs, DELETEs to total rows or LOBs since last full image copy. . . : (CRChangesPct)	(10)	
Ratio, as percent, of INSERTs, UPDATEs, DELETEs to total rows or LOBs since last incremental image copy . . . . . : (ICRChangesPct)	(1)	
Ratio, as percent, of INSERTs, UPDATEs, DELETEs to total rows or LOBs since last REORG . . . . . : (RRTInsDelUpdPct)	(20)	
Ratio, as percent, of unclustered INSERTs to total rows or LOBs. . . . . : (RRTUnclustInsPct)	(10)	
Ratio, as percent, of imperfectly chunked LOBs to total rows or LOBS . . . . . : (RRTDisorgLOBPct)	(10)	
Ratio, as percent, of overflow records to total of rows or LOBs since last REORG or LOAD REPLACE . : (RRTIndRefLimit)	(10)	
Limit, number of mass deletes or dropped tables since last REORG or LOAD REPLACE . . . . . : (RRTMassDelLimit)	(0)	
Ratio, as percent, of INSERTs, UPDATEs, DELETEs to total rows or LOBs since last RUNSTATS. . . . . : (RRTRunStatPct)	(20)	

```

total rows or LOBS . . . . . :      (10)
(RRTDisorgLOBPct)

Ratio, as percent, of overflow records to total
of rows or LOBs since last REORG or LOAD REPLACE . :      (10)
(RRTIndRefLimit)

Limit, number of mass deletes or dropped tables
since last REORG or LOAD REPLACE . . . . . :      (0)
(RRTMassDeLLimit)

Ratio, as percent, of INSERTs, UPDATEs, DELETEs
to total rows or LOBs since last RUNSTATS. . . . . :      (20)
(SRTInsDeLUpdPct)

Limit, sum of INSERTs, UPDATEs, DELETEs since
last RUNSTATS. . . . . :      (0)
(SRTInsDeLUpdAbs)

Limit, number of mass deletes since last REORG
or LOAD REPLACE. . . . . :      (0)
(SRTMassDeLLimit)

```

---

Once you decide what criteria to use and press Enter, you are presented with another screen that provides you with a list of objects that satisfy the stated criteria. You can then choose to run the maintenance that generates a REORG, or a COPY, and so on. Notice that the recommendations are in the last four columns in Example A-14.

*Example A-14 Option 14: Table Space Maintenance*

```

ADB2314 n ----- DB9A Table Space Maintenance ----- Row 1 to 22 of 3,591
Command ==>                                         Scroll ==> PAGE

```

```

Commands:      C - Full Copy  CI - Inc Copy  O - Reorg  R - Runstats
Line commands: C - Full Copy  CI - Inc Copy  O - Reorg  R - Runstats
              AL - Resize

```

Sel	TSname	DBname	Part	Space(KB)	Pct	Num	<---Recommendations--->			
					Used	Ext	Copy	Reorg	Runst	Resize
*	*	*	*		*	*	*	*	*	*
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----
	DSN8S91P	DSN8D91A	0	192	100	1	FUL	YES	NO	NO
	DSN8S91R	DSN8D91A	0	1488	100	2	FUL	YES	YES	NO
	DSN8S91Q	DSN8D91P	0	192	100	1	FUL	NO	YES	NO
	XCUS0001	DSN8D91X	1	720	100	1	FUL	NO	YES	NO
	XCAT0000	DSN8D91X	1	720	100	1	FUL	NO	YES	NO
	XSUP0000	DSN8D91X	1	720	100	1	FUL	NO	YES	NO
	SABITS2	SABIDB	0	720	100	1	FUL	YES	YES	NO
	SABIXML6	DSN00005	1	720	100	1	FUL	YES	YES	NO
	XSAB0000	DSN00005	1	720	100	1	FUL	YES	YES	NO
	SABIXML5	DSN00003	1	720	1	1	FUL	YES	YES	NO
	TESTUNL	DSN00006	1	720	100	1	FUL	YES	YES	NO
	SABIXML9	DSN00007	1	720	100	1	FUL	YES	YES	NO
	XSAB0000	DSN00007	1	720	100	1	FUL	YES	YES	NO
	T1TS	T1DB	0	720	7	1	FUL	NO	YES	NO
	EMP1	DSN00043	1	720	100	1	FUL	YES	YES	NO
	DSN8L91X	DSN8D91A	0	7744	100	1	FUL	NO	YES	NO



DSN8S91S	DSN8D91A	0	48	100	1	FUL	YES	YES	NO
CHCKPAGE	PAOLOR6	0	5040	100	1	FUL	YES	YES	NO
RENAME1	DSNDB04	0	720	100	1	FUL	YES	YES	NO
TB1	DSN00046	1	720	100	1	FUL	NO	YES	NO
TB2	DSN00047	1	720	100	1	FUL	NO	YES	NO
TB3	DSN00048	1	720	100	1	FUL	NO	YES	NO

### **Performance query 14X**

There is a similar screen for indexes, which is illustrated in Example A-15.

#### **Example A-15 Performance query option 14X: parameters for RTS Index Space Maintenance**

ADB2314I ----- DB9A Input Parameters for Real-Time Statistics ----- 20:11

Option ==>

Top of data

The input values specified below are used in the calculations which determine the recommended index space actions. For a full description of any parameter, use panel HELP and refer to the entry indicated by the parenthesized keyword.

Run using default settings: YES (Yes/No)	(default)
	More: +
Limit, number of physical extents. . . . . : (ExtentLimit)	(50)
Limit, number of days since last image copy. . . . . : (CRDaySncLastCopy)	(7)
Ratio, as percent, of updated pages to preformatted pages. . . . . : (CRUpdatedPagesPct)	(1)
Ratio, as percent, of INSERTs, UPDATEs, DELETEs to total rows or LOBs since last image copy. . . . . : (CRChangesPct)	(10)
Limit, number of active pages. . . . . : (CRIndexSize)	(50)
Ratio, as percent, of sum of inserted and deleted index entries to total since last REORG. . . . . : (RRIInsertDeletePct)	(20)
Ratio, as percent, of inserted index entries with key greater than max to total since last REORG, key greater than max to total since last REORG, REBUILD INDEX or LOAD REPLACE. . . . . : (RRIAppendInsertPct)	(10)
Ratio, as percent, of pseudo-deleted index entries to total since last REORG, REBUILD INDEX or LOAD REPLACE . . . . . : (RRIPseudoDeletePct)	(10)
Limit, number of mass deletes since last REORG, REBUILD, or LOAD REPLACE . . . . . : (0)	(0)

(RRIMassDeLimit)

Ratio, as percent, of number of index page splits  
far from original to total since last REORG,  
REBUILD INDEX or LOAD REPLACE. . . . . : (10)  
(RRILeafLimit)

Limit, number of added or removed levels in index  
tree since last REORG, REBUILD INDEX, or LOAD  
REPLACE. . . . . : (0)  
(RRINumLevelsLimit)

Ratio, as percent, of number of index page splits  
far from original to total since last REORG,  
REBUILD INDEX or LOAD REPLACE. . . . . : (10)  
(RRILeafLimit)

Limit, number of added or removed levels in index  
tree since last REORG, REBUILD INDEX, or LOAD  
REPLACE. . . . . : (0)  
(RRINumLevelsLimit)

Ratio, as percent, of number of inserted and deleted  
index entries to total since last RUNSTATS . . . . : (20)  
(SRIInsDelUpdPct)

Limit, number of inserted and deleted index entries  
since last RUNSTATS. . . . . : (0)  
(SRIInsDelUpdAbs)

Limit, number of mass deletes since last REORG,  
REBUILD INDEX or LOAD REPLACE. . . . . : (0)  
(SRIMassDeLimit)

Pressing Enter gives you the results in Example A-16, which are similar to those for table spaces.

*Example A-16 Option 14X: Index Space maintenance*

---

ADB2314X	-----	DB9A	Index Space Maintenance	-----	Row 1 to 2 of 2
Command ==>					Scroll ==> PAGE
Commands: C - Copy O - Reorg R - Runstats					
Line commands: C - Copy O - Reorg R - Runstats AL - Resize					
	Index				<---Recommendations--->
Sel	Space	DBname	Part Nactive	Space	Ext Copy Reorg Runst Resize
*	*	*	*	*	* * * * *
-----					
	DEPTABCD	DSNDB04	0	180	720 1 N/A YES YES NO
	DSNR10KT	DSNDB04	0	180	720 1 N/A YES YES NO
***** END OF DB2 DATA *****					

---

Although it is easy to use the RTS maintenance screens for table spaces or index spaces, it is still a somewhat *ad hoc* or manual process. If you are looking for a tool that does this in an automated fashion, then you want to take a look at the DB2 Automation Tool information found in Appendix A.3, “DB2 Automation Tool” on page 435.

## A.2 DB2 Utility Enhancement Tool for z/OS

The DB2 Utility Enhancement Tool (UET) for z/OS provides additional and complementary functionality to the DB2 Utility Suite for REORG, LOAD, and CHECK. There is both an ISPF user interface as well as a batch component that contains a DSNUTILB intercept. The ISPF user interface allows you to manage thread activity. DB2 UET allows you to

- ▶ Cancel or block threads on DB2 objects to allow utility processing to complete
- ▶ Prevents users from accessing an object when a utility executes
- ▶ Run the utility
- ▶ Change an object’s status to allow users to access it

For more information about DB2 UET, go to the following address:

<http://www.ibm.com/software/data/db2imstools/db2tools/db2-utilities-enhancement/>.

### A.2.1 ISPF user interface

The ISPF user interface allows you to display and cancel threads. You can display a filtered list of active threads and selectively cancel threads.

### A.2.2 Batch processing

You may want to cancel or block threads that interfere with utility processing. For example, you may want to run an online REORG, but after all the different phases complete, and you are waiting for the switch phase to end, there may be some threads that prevent the utility from completing successfully.

You create a batch job step that allows you to include requests for different threads associated with particular utilities to either block or cancel threads. You may need to block new threads on these objects until a specific utility completes or cancel certain active threads that are preventing DB2 online utilities from completing. You can specify which threads are to be canceled by defining certain criteria. The criteria includes a set of objects that can include object name, owner, and so on. Masking is also available.

There is a DSNUTILB intercept program provided by DB2 UET that implements these features. Basically, DB2 UET reads the input file and makes any necessary modifications before passing the modified syntax to DSNUTILB.

### A.2.3 REORG utility options

The main feature available with DB2 UET in conjunction with the REORG utility is the mapping table and index. We discuss this feature in this section.

## Mapping table and index

The DB2 UET can dynamically create the mapping table and its index, which are necessary for REORG SHRLEVEL CHANGE. It can also delete these objects when the online REORG completes. The REORG SHRLEVEL CHANGE JCL need not be modified at all, as there are options to ignore the mapping table specification if these options are provided in the JCL.

Here is an example of a batch online reorganization utility job. As you can see from the syntax, in this case, the utility control statement does not specify a mapping table, which is required for a REORG SHRLEVEL CHANGE. The DB2 UET will create the mapping table and insert the correct syntax into the control statement before passing it off to DB2:

```
REORG      TABLESPACE MTSTBO.MTSTBO
           COPYDDN(TSYSCOPY)
           SHRLEVEL CHANGE
           MAXRO 20 DRAIN ALL
           DRAIN_WAIT 20 RETRY 120 RETRY_DELAY 60 TIMEOUT TERM
           WORKDDN(TSYSUT1,TSORTOUT)
           SORTDEVT 3390
           STATISTICS TABLE(ALL) INDEX(ALL KEYCARD)
```

Example A-17 shows a part of the output from the REORG utility. It shows that the DSNUTILB intercept program was able to allocate the necessary mapping table and index before passing the control statement off to DB2. The output also shows the original REORG control statement without the MAPPINGTABLE option, and the utility control statement after DB2 UET's modification added the MAPPINGTABLE option. These are all highlighted in bold in Example A-17.

### Example: A-17 Online REORG output using DB2 UET

---

```
ABPU5001I 288 11:01:54.06 IBM DB2 Utilities Enhancement Tool Version 0210, FMID=H2AM210,COMP_ID=5655-T58
ABPU5012I 288 11:01:54.06 Connected to started task ABPID=ABP1
ABPU5002I 288 11:01:54.11 Initialization is complete.
-----
ABPU5004I 288 11:01:55.86 Analysis started. Step=1
ABPU5005I 288 11:01:55.86 Analysis completed. RC=00000000
ABPU5006I 288 11:01:55.86 Thread cancel started. Step=1
ABPU5007I 288 11:01:58.35 Thread cancel completed. RC=0000001C
1DSNU000I 288 11:01:59.23 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = MARY00
DSNU1044I 288 11:01:59.27 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
DSNU050I 288 11:01:59.28 DSNUGUTC - EXEC SQL CREATE TABLE DB2R8."MARY00" (TYPE
CHAR( 01 ) NOT NULL, SOURCE_RID CHAR( 05 ) NOT NULL, TARGET_XRID CHAR( 09 ) NOT NULL, LRSN
CHAR( 06 ) NOT NULL) IN ABPGG01T.ABPMAPTS CCSID EBCDIC ENDEXEC
DSNU1180I 288 11:01:59.34 DSNUGSQL - SQLCODE = 000, SUCCESSFUL EXECUTION
DSNU050I 288 11:01:59.34 DSNUGUTC - EXEC SQL CREATE UNIQUE INDEX DB2R8."MARY00 IX" ON DB2R8."MARY00"
(SOURCE_RID ASC, TYPE, TARGET_XRID, LRSN) USING STOGROUP SYSDEFLT
PRIQTY 00000012 SECQTY 00000001 ERASE NO BUFFERPOOL BPO_CLOSE NO ENDEXEC
DSNU1180I 288 11:01:59.46 DSNUGSQL - SQLCODE = 000, SUCCESSFUL EXECUTION
DSNU010I 288 11:01:59.46 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
ABPU5008I 288 11:02:00.14 Utility execution started. Step=1
ABPU5330I 288 11:02:00.14 Original DSNUTILB syntax follows:
ABPU5331I 288 11:02:00.14 REORG TABLESPACE MTSTBO.MTSTBO COPYDDN(TSYSCOPY) SHRLEVEL CHANGE MAXRO 20 DRAIN
ABPU5331I 288 11:02:00.14 ALL DRAIN_WAIT 20 RETRY 120 RETRY_DELAY 60 TIMEOUT TERM WORKDDN (TSYSUT1,
ABPU5331I 288 11:02:00.14 TSORTOUT) SORTDEVT 3390 STATISTICS TABLE(ALL) INDEX(ALL KEYCARD)
ABPU5332I 288 11:02:00.14 End of original DSNUTILB syntax listing.
1DSNU000I 288 11:02:00.26 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = MARY00
DSNU1044I 288 11:02:00.31 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
ODSNU050I 288 11:02:00.31 DSNUGUTC - TEMPLATE TSORTOUT DSN('&US..&SS..&DB..&SN..S&JU (3,5)..#&TI.')
```

```

DSNU050I   288 11:02:00.31 DSNUGUTC - TEMPLATE TSYSUT1 DSN('&US.&SS.&DB.&SN..U&JU(3,5)..#&TI.')
DISP(MOD,DELETE, CATLG)
DSNU1035I  288 11:02:00.31 DSNUJTDR - TEMPLATE STATEMENT PROCESSED SUCCESSFULLY
ODSNU050I  288 11:02:00.31 DSNUGUTC - TEMPLATE TSYSCOPY DSN('&US.&SS.&DB.&SN.&IC.&JU(3,5)..#&TI.')
SPACE TRK MAXPRIME 65536 DISP(MOD, CATLG, CATLG)
DSNU1035I  288 11:02:00.31 DSNUJTDR - TEMPLATE STATEMENT PROCESSED SUCCESSFULLY
DSNU050I   288 11:02:00.31 DSNUGUTC - REORG TABLESPACE MTSTBO.MTSTBO COPYDDN(TSYSCOPY) SHRLEVEL CHANGE
MAXRO 20 DRAIN ALL DRAIN_WAIT 20 RETRY 120 RETRY_DELAY 60 TIMEOUT TERM WORKDDN(TSYSUT1, TSORTOUT)
SORTDEVT 3390 STATISTICS TABLE(ALL) INDEX(ALL KEYCARD) MAPPINGTABLE DB2R8."MARY00"
DSNU1038I  288 11:02:05.93 DSNUGDYN - DATASET ALLOCATED.  TEMPLATE=TSYSCOPY
                        DDNAME=SYS00013
                        DSN=DB2R8.DB9A.MTSTBO.MTSTBO.F09288.#150224
DSNU3340I  288 11:02:05.94 DSNURPCT - UTILITY PERFORMS DYNAMIC ALLOCATION OF SORT DISK SPACE
DSNU3342I  288 11:02:06.34 DSNURPCT - NUMBER OF OPTIMAL SORT TASKS = 31, NUMBER OF ACTIVE SORT TASKS = 4
DSNU1160I  288 11:02:06.36 DSNURPRD - PARTITIONS WILL BE UNLOADED/RELOADED IN PARALLEL, NUMBER OF TASKS =
2
DSNU395I   288 11:02:06.36 DSNURPRD - INDEXES WILL BE BUILT IN PARALLEL, NUMBER OF TASKS = 6
DSNU397I   288 11:02:06.36 DSNURPRD - NUMBER OF TASKS CONSTRAINED BY CPUS DSNU251I  -DB9A 288 11:02:14.93
DSNURPUT - UNLOAD PHASE STATISTICS - NUMBER OF RECORDS UNLOADED=160767 FOR TABLESPACE MTSTBO.MTSTBO PART 1
DSNU251I  -DB9A 288 11:02:14.93 DSNURPUT - UNLOAD PHASE STATISTICS - NUMBER OF RECORDS UNLOADED=224 FOR
TABLESPACE MTSTBO.MTSTBO PART 2
DSNU251I  -DB9A 288 11:02:14.93 DSNURPUT - UNLOAD PHASE STATISTICS - NUMBER OF RECORDS UNLOADED=2982 FOR
TABLESPACE MTSTBO.MTSTBO PART 25
.
.
.
DSNU250I   288 11:02:14.93 DSNURPRD - UNLOAD PHASE COMPLETE, ELAPSED TIME=00:00:08
DSNU303I  -DB9A 288 11:02:31.63 DSNURWT - (RE)LOAD PHASE STATISTICS - NUMBER OF RECORDS=160767 FOR TABLE
S.MTSTBO PART=1
DSNU303I  -DB9A 288 11:02:31.63 DSNURWT - (RE)LOAD PHASE STATISTICS - NUMBER OF RECORDS=224 FOR TABLE
S.MTSTBO PART=2
DSNU393I  -DB9A 288 11:02:39.39 DSNURBXA - SORTBLD PHASE STATISTICS - NUMBER OF KEYS=47541
FOR INDEX S.I_MTSTBO_4 PART 25
DSNU394I  -DB9A 288 11:02:41.32 DSNURBXA - SORTBLD PHASE STATISTICS - NUMBER OF KEYS=1248796
FOR INDEX S.I_MTSTBO_5
DSNU391I   288 11:02:41.50 DSNURPTB - SORTBLD PHASE STATISTICS. NUMBER OF INDEXES = 6
DSNU392I   288 11:02:41.50 DSNURPTB - SORTBLD PHASE COMPLETE, ELAPSED TIME = 00:00:09
DSNU1162I  288 11:02:41.50 DSNURLGD - LOG APPLIES WILL BE PERFORMED IN PARALLEL, NUMBER OF TASKS = 2
DSNU386I   288 11:02:42.84 DSNURLGD - LOG PHASE STATISTICS. NUMBER OF ITERATIONS = 1,
NUMBER OF LOG RECORDS = 0
DSNU385I   288 11:02:42.84 DSNURLGD - LOG PHASE COMPLETE, ELAPSED TIME = 00:00:01
DSNU400I   288 11:02:42.86 DSNURBID - COPY PROCESSED FOR TABLESPACE MTSTBO.MTSTBO
                        NUMBER OF PAGES=54428
                        AVERAGE PERCENT FREE SPACE PER PAGE = 6.01
                        PERCENT OF CHANGED PAGES =100.00
                        ELAPSED TIME=00:00:36
DSNU387I   288 11:02:44.48 DSNURSWT - SWITCH PHASE COMPLETE, ELAPSED TIME = 00:00:01
DSNU428I   288 11:02:44.48 DSNURSWT - DB2 IMAGE COPY SUCCESSFUL FOR TABLESPACE MTSTBO.MTSTBO
DSNU610I  -DB9A 288 11:02:46.19 DSNUSUTP - SYSTABLEPART CATALOG UPDATE FOR MTSTBO.MTSTBO SUCCESSFUL
DSNU610I  -DB9A 288 11:02:46.19 DSNUSUPT - SYSTABSTATS CATALOG UPDATE FOR S.MTSTBO SUCCESSFUL
DSNU610I  -DB9A 288 11:02:46.23 DSNUSUPC - SYSCOLSTATS CATALOG UPDATE FOR S.MTSTBO SUCCESSFUL
DSNU610I  -DB9A 288 11:02:46.23 DSNUSUTB - SYSTABLES CATALOG UPDATE FOR S.MTSTBO SUCCESSFUL
DSNU610I  -DB9A 288 11:02:46.27 DSNUSUCO - SYSCOLUMNS CATALOG UPDATE FOR S.MTSTBO SUCCESSFUL
DSNU610I  -DB9A 288 11:02:46.27 DSNUSUTS - SYSTABLESPACE CATALOG UPDATE FOR MTSTBO.MTSTBO SUCCESSFUL
DSNU610I  -DB9A 288 11:02:47.17 DSNUSUIP - SYSINDEXPART CATALOG UPDATE FOR S.I_MTSTBO_4 SUCCESSFUL
DSNU610I  -DB9A 288 11:02:47.20 DSNUSUPI - SYSINDEXSTATS CATALOG UPDATE FOR S.I_MTSTBO_4 SUCCESSFUL
.
.
.
DSNU610I  -DB9A 288 11:02:47.39 DSNUSUCD - SYSCOLDIST CATALOG UPDATE FOR S.I_MTSTBO_5 SUCCESSFUL

```

```

DSNU620I  -DB9A 288 11:02:47.40 DSNUSEOF - RUNSTATS CATALOG TIMESTAMP = 2009-10-15-11.02.15.692208
DSNU010I   288 11:02:50.84 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
ABPU5009I 288 11:02:51.46 Utility execution completed. SYS=0000, USR=0000
ABPU5003I 288 11:02:51.46 DB2 Utilities Enhancement Tool intercept completed.

```

---

## A.2.4 Load utility options

DB2 UET extends the native LOAD utility syntax to include additional options that are extremely useful. Each of these options provide additional syntax options that can be added to the LOAD utility for performance and additional functionality. The DSNUTILB interface handles these keywords outside of the DB2 LOAD utility and subsequently passes the correct LOAD control statement to DB2 to avoid any syntax errors from DB2.

These options are discussed in this section.

### PRESORT

The PRESORT option sorts rows by table object identifier (OBID) and by clustering index key. If there is no clustering index key available, it sort rows by the oldest defined index. Having the data presorted for LOAD may result in a huge performance boost.

An example of the SYSPRINT using the PRESORT option is provided in Example A-18. When DB2 UET sees the PRESORT option, it will sort the incoming data for LOAD in clustering index key order. Notice also that when it passes the control statement to DB2, it removes the PRESORT option and adds the SORTKEYS 7 parameter, because it knows the exact number of rows to be loaded from the sort process.

#### Example: A-18 SYSPRINT for PRESORT option

---

```

ABPU5004I 296 12:54:07.48 Analysis started. Step=1
ABPU5005I 296 12:54:07.48 Analysis completed. RC=00000000
ABPU5006I 296 12:54:07.48 Thread cancel started. Step=1
ABPU5007I 296 12:54:08.10 Thread cancel completed. RC=00000004
ABPU5008I 296 12:54:08.55 Utility execution started. Step=1
ABPU5330I 296 12:54:08.55 Original DSNUTILB syntax follows:
ABPU5331I 296 12:54:08.55 LOAD PRESORT DATA LOG NO RESUME YES INDDN SYSRECOO INTO TABLE ABPTTEST.
ABPU5331I 296 12:54:08.55 PHYSICAL_VIEW ( PVIEW_ID POSITION( 1 ) CHAR( 4 ) , VIEW_NAME POSITION( 5 ) CHAR(
ABPU5331I 296 12:54:08.55 18) CONSTANT VALUE = '123456789012345678' , VIEW_CREATOR POSITION( 23 ) CHAR(
ABPU5331I 296 12:54:08.55 8) CONSTANT VALUE = '1234' , SECTION_NAME POSITION( 31 ) CHAR( 8) CONSTANT
ABPU5331I 296 12:54:08.55 VALUE = '123456789012345678' , VIEW_DESC POSITION( 39 ) VARCHAR , ARB_CICS_TMS
ABPU5331I 296 12:54:08.55 POSITION( 101: 108) DECIMAL , SHADOW_TABLE_NAME POSITION( 109 ) CHAR( 18) ,
ABPU5331I 296 12:54:08.55 SHADOW_EXPIRE_LIM POSITION( 127 ) SMALLINT , EXIT_PGM_NAME POSITION( 129 ) CHAR(
ABPU5331I 296 12:54:08.55 ( 8) , )
ABPU5332I 296 12:54:08.55 End of original DSNUTILB syntax listing.
1DSNU000I   296 12:54:08.68 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = ABPLOAD
DSNU1044I   296 12:54:08.72 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
ODSNU050I   296 12:54:08.72 DSNUGUTC - LOAD DATA LOG NO RESUME YES
DSNU650I   ) 296 12:54:08.73 DSNURWI - INTO TABLE ABPTTEST.PHYSICAL_VIEW
DSNU650I   ) 296 12:54:08.73 DSNURWI - (PVIEW_ID POSITION(1) CHAR(4),
DSNU650I   ) 296 12:54:08.73 DSNURWI - VIEW_NAME POSITION(5) CHAR(18),
DSNU650I   ) 296 12:54:08.73 DSNURWI - VIEW_CREATOR POSITION(23) CHAR(8),
DSNU650I   ) 296 12:54:08.73 DSNURWI - SECTION_NAME POSITION(31) CHAR(8),
DSNU650I   ) 296 12:54:08.73 DSNURWI - VIEW_DESC POSITION(39) VARCHAR,
DSNU650I   ) 296 12:54:08.73 DSNURWI - ARB_CICS_TMS POSITION(101:108) DECIMAL,
DSNU650I   ) 296 12:54:08.73 DSNURWI - SHADOW_TABLE_NAME POSITION(109) CHAR(18),
DSNU650I   ) 296 12:54:08.73 DSNURWI - SHADOW_EXPIRE_LIM POSITION(127) SMALLINT,
DSNU650I   ) 296 12:54:08.73 DSNURWI - EXIT_PGM_NAME POSITION(129) CHAR(8)), INDDN ABPREC SORTKEYS 7
DSNU320I   ) 296 12:54:08.84 DSNURWI - RESUME(YES) WAS SPECIFIED FOR EMPTY TABLESPACE

```

```

DSNU304I ) 296 12:54:09.12 DSNURWT - (RE)LOAD PHASE STATISTICS - NUMBER OF RECORDS=7 FOR TABLE
ABPTEST.PHYSICAL_VIEW
DSNU1147I ) 296 12:54:09.12 DSNURWT - (RE)LOAD PHASE STATISTICS - TOTAL NUMBER OF RECORDS LOADED=7 FOR
TABLESPACE
ABPDDB.ABPSPVTS
DSNU302I 296 12:54:09.12 DSNURILD - (RE)LOAD PHASE STATISTICS - NUMBER OF INPUT RECORDS PROCESSED=7
DSNU300I 296 12:54:09.12 DSNURILD - (RE)LOAD PHASE COMPLETE, ELAPSED TIME=00:00:00
DSNU042I 296 12:54:09.12 DSNUGSOR - SORT PHASE STATISTICS -
NUMBER OF RECORDS=7
ELAPSED TIME=00:00:00
DSNU349I ) 296 12:54:09.31 DSNURBXA - BUILD PHASE STATISTICS - NUMBER OF KEYS=7 FOR INDEX
ABPTEST.PHYSICAL_VIEW_PX
DSNU258I 296 12:54:09.32 DSNURBXD - BUILD PHASE STATISTICS - NUMBER OF INDEXES=1
DSNU259I 296 12:54:09.32 DSNURBXD - BUILD PHASE COMPLETE, ELAPSED TIME=00:00:00
DSNU381I ) 296 12:54:09.34 DSNUGSRX - TABLESPACE ABPDDB.ABPSPVTS IS IN COPY PENDING
DSNU010I 296 12:54:09.35 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=4
ABPU5009I 296 12:54:17.46 Utility execution completed. SYS=0000, USR=0004
ABPU5010I 296 12:54:17.46 Allow threads started. Step=1
ABPU5011I 296 12:54:17.87 Allow threads completed. RC=00000000

```

---

## CONSTANT and VALUEIF

DB2 UET provides the LOAD utility with both **CONSTANT** and **VALUEIF** options. For the **CONSTANT** value, DB2 UET replaces a value for a particular column with a constant value. For the **VALUEIF** option, you can add a constant value depending on a condition.

Example A-19 shows an example of using the **CONSTANT** option. In this example, the following columns result in constant values:

- ▶ **VIEW\_NAME** will contain the constant value '123456789012345678'.
- ▶ **VIEW\_CREATOR** will contain the constant value '1234'.
- ▶ **SECTION\_NAME** will contain the constant value '123456789012345678'.

### Example A-19 PRESORT and CONSTANT options

```

LOAD PRESORT DATA LOG NO INDDN SYSRECOO INTO TABLE
ABPTEST.PHYSICAL_VIEW
(
  PVIEW_ID                POSITION(      1      )
  CHAR(                     4) ,
  VIEW_NAME                POSITION(      5      )
  CHAR(                     18)
  CONSTANT VALUE = '123456789012345678' ,
  VIEW_CREATOR             POSITION(     23      )
  CHAR(                     8)
  CONSTANT VALUE = '1234' ,
  SECTION_NAME             POSITION(     31      )
  CHAR(                     8)
  CONSTANT VALUE = '123456789012345678' ,
  VIEW_DESC                POSITION(     39      )
  VARCHAR                  ,
  ARB_CICS_TMS             POSITION(    101:    108)
  DECIMAL                  ,
  SHADOW_TABLE_NAME        POSITION(    109      )
  CHAR(                     18) ,
  SHADOW_EXPIRE_LIM        POSITION(    127      )
  SMALLINT                 ,
  EXIT_PGM_NAME            POSITION(    129      )
  CHAR(                     8) ,
)

```

---

Example A-20 shows an example of the VALUEIF option. In this example, the VIEW\_NAME column in position 5 of the row is replaced by the constant value '123456' if the PVIEW\_ID = 'SDLR'.

*Example A-20 VALUEIF option*

---

```

LOAD DATA LOG YES RESUME YES INDDN SYSREC00 INTO TABLE
  ABPTEST.PHYSICAL_VIEW
  (
    PVIEW_ID                                POSITION(      1      )
    CHAR(                                     4) ,
    VIEW_NAME POSITION(5) CHAR(18) VALUEIF (PVIEW_ID) = 'SDLR'
    VALUE = '123456' ,
    VIEW_CREATOR                            POSITION(     23      )
    CHAR(                                    8) ,
    SECTION_NAME                            POSITION(     31      )
    CHAR(                                    8) ,
    VIEW_DESC                               POSITION(     39      )
    VARCHAR                                ,
    ARB_CICS_TMS                            POSITION(    101:    108)
    DECIMAL                                ,
    SHADOW_TABLE_NAME                       POSITION(     109      )
    CHAR(                                   18) ,
    SHADOW_EXPIRE_LIM                       POSITION(     127      )
    SMALLINT                               ,
    EXIT_PGM_NAME                           POSITION(     129      )
    CHAR(                                    8)
  )
LOAD DATA LOG YES RESUME YES INDDN SYSREC01 INTO TABLE
  ABPTEST.LOGICAL_VIEW
  (
    LVIEW_ID                                POSITION(      1      )
    CHAR(                                     4) ,
    DOCUMENT_NAME POSITION(5) CHAR(8) VALUEIF (LVIEW_ID) = 'V001'
    VALUE = '1234' ,
    ROOT_LVIEW                              POSITION(     13      )
    CHAR(                                    1) ,
    MANDATORY_LVIEW                         POSITION(     14      )
    CHAR(                                    1) ,
    COMPLETE_LVIEW_POS                      POSITION(     15      )
    SMALLINT                               ,
    LVIEW_DESC                              POSITION(     17      )
    VARCHAR                                ,
    ARB_CICS_TMS                            POSITION(     79:     86)
    DECIMAL                                ,
    ASCA_CERTIFICATION                      POSITION(     87      )
    CHAR(                                    1)
  )

```

---



## A.2.5 CHECK DATA utility

DB2 UET extends the CHECK DATA utility to allow you to write discarded rows to a sequential file. The syntax includes the parameters DISCARDTO and DISCARDSPACE as an alternative to the USE table\_name2 clause. Both discarded rows and LOAD utility control statements are written to a file that can optionally be managed using TEMPLATE statements.

**Important:** The LOAD and CHECK DATA syntax requires manual changes, whereas the REORG syntax does not require any manual changes.

## A.3 DB2 Automation Tool

The DB2 Automation Tool is a powerful productivity aid for automatic utility generation. It allows you to generate DB2 utility JCL against a predefined set of DB2 objects. Once you have these objects specified in an object profile, you need to set up a utility profile in which you describe all of the specific utility parameters. Optionally, an exception profile allows you to define specific criteria that is used to determine if the utility JCL should be generated. These exceptions can be from many different and varied sources, such as statistics from the DB2 catalog, RTS, a particular day of the week, and much more. Finally, you define a job profile that points to a specific object, utility and exception profile. In addition, the job profile contains additional information about the utility job(s) that are generated. There are many optional criterion to choose from, including but not limited to the number of utility steps per job, the number of jobs, the actual jobname to be used, and so on.

In this section, we give you a brief overview of some of the highlights of this tool.

First, we begin with the DB2 Automation Tool main menu, which is shown in Example A-21.

*Example A-21 DB2 Automation Tool main menu*

```
HAA$MAIN V3R1 ----- IBM DB2 Automation Tool for z/OS ---- 2009/10/15 19:43:04
Option  ==>
```

```
-----
Commands: END - Return to ISPF.
```

```
Options: 0 - Setup              7 - DB2 Command Processor
          1 - Object Profiles    8 - Dataset Manager
          2 - Utility Profiles   9 - Data Page Display
          3 - Exception Profiles 10 - Disaster Recovery
          4 - Job Profiles       11 - Stand Alone Utilities
          5 - Quick Build        12 - Exit
          6 - Execution Reports
```

```
-----
DB2 Subsystem ID: DB9A          (1-4 Character Subsystem ID or ? for list)
Current SQLID:                  User: DB2R8 - HAA
-----
```

For more information about the DB2 Automation Tool, go to the following address:

<http://www.ibm.com/software/data/db2imstools/db2tools/db2-automation-tool/>

### A.3.1 Object profile

An object profile merely specifies the objects to be used in generating the utility. Wildcarding is allowed, which gives you the flexibility to not worry when new table spaces are added to a database. Once you have a utility profile set up, it has the option to be used by other users.

You can add table spaces, indexes, and volumes, and you can specify a wildcard in each one. In Example A-22, we specify that we want to add table spaces.

To define an object profile, you start at the main menu (shown in Example A-21 on page 435) using option '1' or 'O'.

*Example A-22 Object profile display*

```
HAAS$OPRU V3R1      ----- Update Object Profile Display ----- 2009/10/14 22:41:06
Option ===>                                     Scroll ===> PAGE
```

```

      Commands: Explode - View all objects. End - Return to previous screen.
Line Commands: A - Add D - Delete E - Explode U - Update R - Repeat
Creator: DB2R8          Profile: ALL OBJECTS                      User: DB2R8
Description: ALL OBJECTS OWNED BY DB2*
Share Option: U (U - Update, V - View, N - No)                    Row 1 of 1    >
-----E ssssssss Add Objects to the Object Profile ssssssssN -----
        e HAA$OPRA                                             e
           Wi e   Add Tablespaces                Y   (Y - Yes, N - No)     e
Cmd Type Ca e                                           e
A Press E e   Add Indexes                        N   (Y - Yes, N - No)     e
*****e                                               e *****
           e   Add Volumes                       N   (Y - Yes, N - No)     e
           e                                              e
           e   Press ENTER to process or PF3 to Cancel       e
           e                                              e
           e                                              e
DssssssssssssssssssssssssssssssssssssssssssssssssssM
```

In Example A-23, we add all of the table spaces owned by DB2R\*. Notice that we are using a wildcard specification, which will allow the tool to include all new table spaces, including those added in the future.

### Example A-23 Adding objects to an Object Profile

```

HAAS$OPRU V3R1      ----- Update Object Profile Display ----- 2009/10/14  22:41:06
Option  ==>                                     Scroll ==> PAGE

```

```

Commands: Explode - View all objects. End - Return to previous screen.
Line Commands: A - Add D - Delete E - Explode U - Update R - Repeat
Creator: DB2R8 Profile: ALL OBJECTS User: DB2R8
Description: ALL OBJECTS OWNED BY DB2*
Share Option: U (U - Update, V - View, N - No) Row 1 of 1 >
E s s s s s s s s s s s s s s s s Enter Tablespaces Like to Display s s s s s s s s s s s s s s s s N
e HAA$OTL9 e
e Commands: ENTER - Continue e
e END - Cancel e
e ----- e
e Database Like. . * Wildcard . Y (Y - Yes, N - No) e
e Tablespace Like. * Exclude. . I (E - Exclude, I - Include) e

```

```

e Creator Like . . DB2R*      >
e
e Process Dependent Indexes . . . . . N (Y - Yes, N - No)
e Process Referentially Dependent Tablespace. N (Y - Yes, N - No,
e                                     B - Build time Expansion,
e                                     R - Run time Expansion)
e Process Cloned Tables. . . . . N (Y - Yes, N - No)
e
e
e
e
DssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssM

```

---

Notice in Example A-24 that you can include a partitioned table space at the table space level or at the partition level. This way you can run a utility at the partition level instead of at the table space level.

*Example A-24 Partitioned table space handling*

---

```

HAA$OPRU V3R1  ----- Update Object Profile Display ----- 2009/10/14 22:41:06
Option  ==>                                           Scroll ==> PAGE

      Commands: Explode - View all objects.  End - Return to previous screen.
Line Commands: A - Add  D - Delete  E - Explode  U - Update  R - Repeat
Creator: DB2R8      Profile: ALL OBJECTS      User: DB2R8
Description: ALL OBJECTS OWNED BY DB2*
Share Option: U  (U - Update, V - View, N - No)      Row 1 of 1  >
----- E s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s N --
e HAA$PART
      Wi e  Utilities can run against each partition or it can
Cmd  Type Ca e  run against all partitions.  When HAA explodes wild
A   Press E e  card table and index spaces, which method would you
***** e  like partitioned spaces exploded?
      e
      e          Explode  P  (A - All, P - Partitioned)
      e
      e
      e
      e
      D s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s s M

```

---

In Example A-25, we illustrate a summary of the current objects in the Object Profile. Because we used the wildcard feature, all objects that belong to creator DB2R\* will be included when a utility is generated.

Example A-25 Summary of objects in an Object Profile

```
HAA$OPRU V3R1 ----- Update Object Profile Display ----- 2009/10/14 22:47:18
Option ==> Scroll ==> PAGE

Commands: Explode - View all objects. End - Return to previous screen.
Line Commands: A - Add D - Delete E - Explode U - Update R - Repeat
Creator: DB2R8 Profile: ALL OBJECTS User: DB2R8
Description: ALL OBJECTS OWNED BY DB2R*
Share Option: U (U - Update, V - View, N - No) Row 1 of 1 >
-----
                                Volume /
                                Wild -- Process -- Inc/ IX DB Name/ IX Crtr/ IX Name/
Cmd  Type Card IX  RI Clone Exc TS Crtr DB Name TS Name
      TS   Y  N   N   N  INC DB2R*  *      *
***** Bottom of Data *****
```

If you want to see the current objects, you can issue the Explode command; its results are shown in Example A-26.

Example A-26 Explode current object list

HAA\$OPRE V3R1										----- Explode Object Profile Display -----										2009/10/14 22:48:03																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																
Option ==>																				Scroll ==> PAGE																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																
Commands: End - Return to the previous screen.																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																				
Line Commands: S - Select to Exclude															Row 1 of 254															+>																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																						
-----																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																				
Creator: DB2R8										Profile: ALL OBJECTS										User: DB2R8																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																
-----																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																				
																				Volume /																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																
										Wild -- Process --										Inc/ IX DB Name/ IX Crtr/ IX Name/																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																
Cmd	Type	Card	IX	RI	Clone	Exc	TS	Crtr	DB Name	TS Name																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																										

## A.3.2 Utility profile

We examine the contents of a utility profile in this section. In Example A-27, we provide a view of the Utility Profile Options screen, where you choose the specific utility associated with this utility profile. Although you can choose more than one utility, it is usually best to have only one utility per utility profile. This way you can order the utilities by assigning an order to the utility profiles when you finally create a job profile. Notice that we are going to choose the COPY utility and look at the specific options for this utility in this section. The other functions that you can include are also seen in Example A-27, and include:

- ▶ Automation tool utility
  - Data page verification reporting: Checks the contents of a page
  - Reallocation: Reallocates the size of the object using a specified formula
- ▶ DB2 utilities
  - Recover: RECOVER utility
  - Image copy: COPY full or incremental
  - Copytocopy: COPYTOCOPY utility
  - Runstats: RUNSTATS utility
  - TS reorg: REORG TABLESPACE
  - IX reorg: REORG INDEX
  - Quiesce: QUIESCE utility
  - Modify: MODIFY utility
  - Repair: REPAIR utility
  - Rebind: Generates a REBIND for all plans or packages associated with the object(s) defined in the object profile

To define a utility profile, start at the main menu shown in Example A-21 on page 435 using option '2' or 'U'.

*Example A-27 Utility profile options*

---

```
HAA$UOPT V3R1 ----- Utility Profile Options ----- 2009/10/14 22:50:29
Option  ===>

      Commands: END - Return to the previous screen.
      Creator: DB2R8      Profile: FIC      User: DB2R8
Share Option ==> U  U - Update      Description: FULL IMAGE COPY
                  V - View
                  N - No

                                --Include in Profile-- -View Utility Options-
Data Page Verification Reporting => N (Y - Yes, N - No) => N (Y - Yes, N - No)
Reallocation                     => N (Y - Yes, N - No) => N (Y - Yes, N - No)
Recover . . . . .                => N (Y - Yes, N - No) => N (Y - Yes, N - No)
Image Copy                      => Y (Y - Yes, N - No) => Y (Y - Yes, N - No)
Copy to Copy . . . . .           => N (Y - Yes, N - No) => N (Y - Yes, N - No)
Runstats                       => N (Y - Yes, N - No) => N (Y - Yes, N - No)
TS Reorg . . . . .              => N (Y - Yes, N - No) => N (Y - Yes, N - No)
IX Reorg                       => N (Y - Yes, N - No) => N (Y - Yes, N - No)
Quiesce . . . . .              => N (Y - Yes, N - No) => N (Y - Yes, N - No)
Modify                          => N (Y - Yes, N - No) => N (Y - Yes, N - No)
Repair . . . . .                => N (Y - Yes, N - No) => N (Y - Yes, N - No)
Rebind                          => N (Y - Yes, N - No) => N (Y - Yes, N - No)
```

---

This is the first screen you see for the Copy utility in Example A-28. Each utility provides a capability to insert a step before or after the utility to accommodate special needs.

*Example A-28 Image copy options*

---

```

HAA$UTSC V3R1 ----- Image Copy options ----- 2009/10/14 22:55:18
Option ==>                                         Scroll ==> PAGE
Commands: END - Return to the previous screen.
Press <PF7/PF8> to scroll for additional options.
Creator: DB2R8      Name: FIC                                User: DB2R8
                                                    More:      +

Exception Rule . . . . . ==> A (A - Accepted, R - Rejected, B - Both)
Image Copy Utility mode      ==> D (D - DB2, S - Symmetrix, E - Ess)
  Alter EMC Symm/IBM ESS Optns ==> N (Y - Yes, N - No)
Alter Image Copy DSN specs   ==> Y (Y - Yes, N - No)
Utility ID . . . . . ==> MARYFIC      (16 characters)

Parallel                    ==> N (Y - Yes, N - No)
  Number of objects . . . . ==>      (0 - 99)
  Number of tape units      ==>      (0 - 99)
Filter DDname . . . . . ==>          (8 character DD name)
Sharelevel                  ==> R (R - Reference, C - Change)
Full Image Copy . . . . . ==> N (Y - Yes, N - No)
Check Page                  ==> N (Y - Yes, N - No)
Concurrent . . . . . ==> N (Y - Yes, N - No)
Change Limit
  First Percent              ==> 20    (% value)

Second Percent . . . . . ==>          (% value)
  Report only                ==> N (Y - Yes, N - No)
  Max Tape Volume/DASD Unit Cnt ==> 5    (1-255 volumes)
  Stack Copy Control Cards . . ==> Y (Y - Yes, N - No)
  Scope                      ==> A (A - All, P - Pending)

Optional Skeletals:          -- BEFORE --    -- AFTER --
JCL Skeletal . . . . . ==>                ==>      (8 Character Name)
Control Cards Skeletal      ==>                ==>      (8 Character Name)
Step End Skeletal . . . . . ==>                ==>      (8 Character Name)

```

---

More options specific to image copy are shown in Example A-29, Example A-30 on page 442, and Example A-31 on page 443.

*Example A-29 Image copy options 2*

```
HAA$UIMG V3R1 ----- Image Copy Options ----- 2009/10/14 22:57:06
Option ==>
  Commands: END - Return to the previous screen.

  Creator: DB2R8      Name: FIC                        User: DB2R8

  Enter the Image Copy options to associate with this utility profile

                                Take Image Copy      View/Update Options

Local Primary . . . . . ==> Y  (Y - Yes,      ==> Y (Y - Yes,
                                N - No)        N - No)
Local Backup      ==> N  (Y - Yes,      ==> N (Y - Yes,
                                N - No)        N - No)
Recovery Site Primary . . . ==> N  (Y - Yes,      ==> N (Y - Yes,
                                N - No)        N - No)
Recovery Site Backup      ==> N  (Y - Yes,      ==> N (Y - Yes,
                                N - No)        N - No)
```

```

HAA$UCPO V3R1 ----- Image Copy Options ----- 2009/10/14 22:57:37
Option      ==>

```

```

Creator: DB2R8      Name: FIC      User: DB2R8
Commands: END - Return to the previous screen.
Press <PF7/PF8> to scroll for additional options.
More:      +

```

Image Copy options for IMAGE COPY LOCAL PRIMARY		
Use Threshold Unit if allocated space exceeds x Meg/Trks/Cyls Optional		
	Std Unit	Threshold Unit
Update DSN create spec	=> Y	=> N (Y - Yes, N - No)
Unit Type	=>	=> (CART - DISK - etc.)
Catalog Options		
DISP=Status . . . . .	=> M	=> M (M - MOD, N - NEW, O - OLD, S - SHR)
Normal Termination	=> C	=> C (C - CATLG, D - DEL, K - KEEP, U - UNCATLG)
Abnormal Termination	=> C	=> C (C - CATLG, D - DEL, K - KEEP, U - UNCATLG)
Data Class . . . . .	=>	=> (8 character class)
Storage Class	=>	=> (8 character class)
Management Class . . . .	=>	=> (8 character class)
	Std Unit	Threshold Unit
Update DSN create spec	=> Y	=> N (Y - Yes, N - No)
Unit Type	=>	=> (CART - DISK - etc.)
Catalog Options		
DISP=Status . . . . .	=> M	=> M (M - MOD, N - NEW, O - OLD, S - SHR)
Normal Termination	=> C	=> C (C - CATLG, D - DEL, K - KEEP, U - UNCATLG)
Abnormal Termination	=> C	=> C (C - CATLG, D - DEL, K - KEEP, U - UNCATLG)
Data Class . . . . .	=>	=> (8 character class)
Storage Class	=>	=> (8 character class)
Management Class . . . .	=>	=> (8 character class)
Tape specific parameters Only needed if Unit Type is a Tape device:		
Expiration date *or*	=>	=> (YYYYDDDD - YYDDDD)
Retention period	=>	=> (4 digit number)



#### Example A-31 Image copy options 4

---

```
HAA$UCPN V3R1 ----- LP Image Copy DSN Generation ----- 2009/10/14 23:01:01
Option ==>
  Commands: END - Return to the previous screen.
  Creator: DB2R8      Name: FIC                      User: DB2R8
  IMAGE COPY LOCAL PRIMARY NON-THRESHOLD
  Qualifier code ==>   Free form literal ==>          Show DSN ==> N
  GDG Limit      ==>   (1-255)
  Current dataset name generation qualifier string:

Valid dataset name generation codes are:
( * marked items are not supported in IC dynamic dataset generation.)
  1. Database          11. Date (YYYYDDD)      21. Unique
  2. Space Name        12. Year (YYYY)         * 22. GDG (+1)..(+n)
  3. Partition/DSNUM   13. Month (MM)          23. ICBACKUP (#24.#25)
* 4. Volser            14. Day (DD)            24. Local/Recovery (L/R)
* 5. Partition/DSNUM   15. Julian Day (DDD)     25. Primary/Backup (P/B)
    only when partitioned 16. Hours (HH)       26. ICTYPE (Full/Incr/Cond)
* 7. Vcatname          17. Minutes (MM)        27. Utility Name
  8. Subsystem ID      18. Seconds (SS)        28. Job Name
* 9. User ID           * 19. Timestamp         29. Step Name
  10. Time (HHMMSS)    * 20. Random Number     30. Substring Qualifier
                                      31. Use freeform literal
```

---

### A.3.3 Exception profile

An exception profile is where you specify the particular conditions that need to be checked when you run the job profile to determine if a utility should be executed. In the exception profile, you set up the different conditions to be checked. If an object satisfies the criteria, then the utility JCL is generated for the object. This section illustrates how to set up the exception criteria.

The first example illustrates the different exception criteria you can specify. In Example A-32 on page 444, you can specify a day of the week or a day of the month. If you page down, you will see DB2 catalog fields and RTS values that can also be used for exception criteria. More options are listed in Table A-1 on page 444.

Here you must select the particular criteria and, if applicable, a condition and a value. For a day of the week, no condition is necessary. For DAY OF MONTH, you specify EQ for the condition and a '1' for the value if you want the utility triggered on the first Monday of the month.

You also have the option to select multiple criteria and AND or OR them together. There is a lot of flexibility here. So you can easily choose to image copy an object if the number of days since the last image copy is more than five or if the object is in COPY PENDING status.

To define an exception profile, start at the main menu shown in Example A-21 on page 435 with option '3' or 'E'.

*Example A-32 Exception profile*

```

HAA$EPRU V3R1  ----- Update Exceptions Profile Display ----- 2009/10/15 17:35:
Option ==>                                           Scroll ==> PAGE
      Commands: END - Save and exit.
Line Commands: A - And  O - Or  S - Select  D - Deselect  R - Repeat
      CONditions: LT|<|LE|<=|EQ|=|GT|>|GE|>=|NE|¬=<> "*" indicates DAT stat
----- Row 1 of 181  +>
Creator: DB2R8      Profile: FIC EXCEPTION      User: DB2R8
Share Option: U (U - Update, V - View, N - No)
Description:                               Scroll Right for Column Help
Use Stats From: R (R - Repository, C - Catalog, U - Runstats,
                  S - Shadow, H - History)
Update Runstats Options: N (Y - Yes, Save Triggers in Repository: N (Y - Yes,
                           N - No)                                N - No)
Combine IX/TS Exceptions if evaluating IX triggering a TS: N (Y - Yes, N - No)
-----
S Statistics Type--- *Column----- Cond -----Exception Value-----
DAY OF WEEK          MONDAY
                     TUESDAY
                     WEDNESDAY
                     THURSDAY
                     FRIDAY
                     SATURDAY
                     SUNDAY
DAY OF MONTH         NTH_MONDAY
                     NTH_TUESDAY
                     NTH_WEDNESDAY
                     NTH_THURSDAY

```

*Table A-1 Exception criteria*

Statistics type	Column name	Cond & exception value
DAY OF WEEK	MONDAY TUESDAY WEDNESDAY THURSDAY FRIDAY SATURDAY SUNDAY	N/A
DAY OF MONTH	NTH_MONDAY NTH_TUESDAY NTH_WEDNESDAY NTH_THURSDAY NTH_FRIDAY NTH_SATURDAY NTH_SUNDAY NTH_DAY LAST_DAY DAY_MONTH	LT, LE, =, EQ, , GT, GE, >=, NE, ¬=, <>
TIME OF DAY	TIME_FROM TIME_TO	EQ

Statistics type	Column name	Cond & exception value
OBJECT	EXCLUDE/ONLY LOB PGSIZE_32K	N/A
USER EXIT	LOAD_MODULE CLIST_REXX_EXEC	<i>name</i>
TABLESPACESTATS REAL TIME STAT	TOTALROWS NACTIVE SPACE EXTENTS	LT, LE, =, EQ, , GT, GE, >=, NE, ¬=, <>
INDEXSPACESTATS REAL TIME STAT	TOTALENTRIES NLEVELS NACTIVE SPACE EXTENTS	LT, LE, =, EQ, , GT, GE, >=, NE, ¬=, <>
REALTIME ICOPY REAL TIME STAT	REORG_OR_LOAD DAYS_SINCE_LAST UPDATED_PAGES UPDATED_PAGES_PCT COPY_CHANGES COPY_CHANGES_PCT	LT, LE, =, EQ, , GT, GE, >=, NE, ¬=, <>
REALTIME REORG TS REAL TIME STAT	DAYS_SINCE_LAST INS_UPD_DEL INS_UPD_DEL_PCT UNCLUST_INS UNCLUST_INS_PCT DISORGED_LOBS DISORGED_LOBS_PCT RELOCATED_ROWS RELOCATED_ROWS_PCT MASS_DELETES	LT, LE, =, EQ, , GT, GE, >=, NE, ¬=, <>
REALTIME REORG IX REAL TIME STAT	DAYS_SINCE_LAST INS_DEL INS_DEL_PCT APPENDED_INS APPENDED_INS_PCT PSEUDO_DEL PSEUDO_DEL_PCT LEAFFAR_SPLITS_PCT NLEAF_SPLITS NLEAF_SPLITS_PCT NUMLEVELS_UPDATED MASS_DELETES	LT, LE, =, EQ, , GT, GE, >=, NE, ¬=, <>
REALTIME RUNSTATS REAL TIME STAT	DAYS_SINCE_LAST INS_UPD_DEL INS_UPD_DEL_PCT MASS_DELETES	LT, LE, =, EQ, , GT, GE, >=, NE, ¬=, <>
MVS CATALOG	*DSNUM *EXTENTS *PERCENT_USED *ALLOCATED_TRACKS *PQTY *SQTY	LT, LE, =, EQ, , GT, GE, >=, NE, ¬=, <>

Statistics type	Column name	Cond & exception value
SYSCOPY	ICTYPE DAYS *CHGD_SINCE_LAST_IC	LT, LE, =, EQ, , GT, GE, >=, NE, ¬=, <>
DB2 DISPLAY STATUS	TRIGGER_IF_1_MATCH STATUS_ARBDP STATUS_AREO* STATUS_AREST STATUS_AUXW STATUS_CHK STATUS_COPY STATUS_GRECP STATUS_ICOPY STATUS_LPL STATUS_LSTOP STATUS_PSRBD STATUS_RBDP STATUS_RBDP* STATUS_RECPC STATUS_REFC STATUS_REORP STATUS_RO STATUS_RW STATUS_RESTP STATUS_STOP STATUS_STOPE STATUS_STOPP STATUS_UT STATUS_UTRO STATUS_UTRW STATUS_UTUT	EQUAL or NOT EQUAL
SYSTABLEPART	CARDF SPACEF PQTY SQTY SECQTYI DSNUM NEARINDREF FARINDREF *TOTINDREF *PERCINDREF PERCACTIVE PERCDROP *DROPSPACE PAGESAVE	LT, LE, =, EQ, , GT, GE, >=, NE, ¬=, <>
SYSTABLESPACE	NACTIVEF PGSIZE	LT, LE, =, EQ, , GT, GE, >=, NE, ¬=, <>
SYSTABLES	ONLY_NONPART_OBJ CARDF NPAGESF SPACEF AVGROWLEN PCTPAGES PCTROWCOMP	LT, LE, =, EQ, , GT, GE, >=, NE, ¬=, <>

Statistics type	Column name	Cond & exception value
SYSTABSTATS	CARDF NPAGES NACTIVE PCTPAGES PCTROWCOMP	LT, LE, =, EQ, , GT, GE, >=, NE, ¬=, <>
SYSCOLUMNS	COLCARDF	LT, LE, =, EQ, , GT, GE, >=, NE, ¬=, <>
SYSCOLSTATS	COLCARD	LT, LE, =, EQ, , GT, GE, >=, NE, ¬=, <>
SYSINDEXES	ONLY_NONPART_OBJ CLUSTERED CLUSTERRATIO CLUSTERRATIOF NLEAF NLEVELS FIRSTKEYCARDF FULLKEYCARDF SPACEF PGSIZE PIECESIZE	LT, LE, =, EQ, , GT, GE, >=, NE, ¬=, <>
SYSINDEXSTATS	CLUSTERRATIO CLUSTERRATIOF NLEAF NLEVELS FIRSTKEYCARDF FULLKEYCARDF KEYCOUNTF	LT, LE, =, EQ, , GT, GE, >=, NE, ¬=, <>
SYSINDEXPART	CARDF SPACEF PQTY SQTY SECQTYI DSNUM LEAFDIST LEAFNEAR LEAFFAR PSEUDO_DEL_ENTRIES NEAROFFPOSF FAROFFPOSF *PERCOFFPOS	LT, LE, =, EQ, , GT, GE, >=, NE, ¬=, <>

Notes for Table A-1 on page 444:

- ▶ Names in the “Column name” beginning with an asterisk ‘\*’ are computed fields.
- ▶ SYSTABSTATS exception conditions ONLY apply to partitioned table spaces.
- ▶ CLUSTERED and CLUSTERRATIOF IX exceptions trigger the associated table space.
- ▶ SYSINDEXSTATS exception conditions ONLY apply to partitioned indexes.
- ▶ CLUSTERRATIOF IX exceptions trigger the associated table space.
- ▶ xxxxOFFPOS exceptions trigger the associated table space.

### A.3.4 Job profile

A job profile is needed to define the objects, the utilities to run, and, optionally, the exceptions to use in the generation of the utilities. In addition, there are many different job generation settings for the sake of flexibility, such as specifying the number of jobs to generate, the number of objects per job, special job names, and so on.

To define a job profile, start at the main menu shown in Example A-21 on page 435 using option '4' or 'J'.

Example A-33 shows the job profile main screen. you can see there are many options. These options may pertain to Setup, Templates, or Listdefs, and Job Break Down options. Refer to Example A-34 on page 449, Example A-35 on page 449, and Example A-36 on page 450 for these special job profile options.

There are also many other options listed on this main job profile screen. For example, you can ask that the utility jobs be generated based on load balancing. The DB2 Automation Tool allows you to build a repository that can remember the actual elapsed time the last time a particular utility ran and used this information in load balancing the generated jobs.

*Example A-33 Job profile display*

```
HAA$JPRD V3R1      ----- Jobs Profile Display ----- 2009/10/15 19:49:26
Option  ===>                               Scroll ===> CSR
```

```

Commands: END          - Cancel and exit. ENTER      - Process new values.
Line Commands: B - Build C - Create D - Delete E - Export
               I - Import R - Rename U - Update V - View
E s s s s s s s s s s s s Generation Options for DB2R8.RUN FICS ON ALL TS s s s s s s s s s s s s N
e HAA$JOPT ==> e
e Commands: END      - Return to Jobs Profile Update. e
e PF7/PF8 - Scroll for additional options. e
e e e
e More: + e
e Update Setup Override Options . . . . . N (Y - Yes, N - No) e
e Update Template/Listdef/Option parms . . N (Y - Yes, N - No) e
e Update Job Break Down Options . . . . . N (Y - Yes, N - No) e
e Automatically Gen GDG Base . . . . . 000 (0-255 Limit) e
e Load Balance jobs by . . . . . N (T - Time, D - Dasd, N - None) e
e Capture run times for Load Balancing . . N (Y - Yes, N - No) e
e Start spaces in Utility/Read Only . . . N (N - No, U - Utility, e
e R - Read only) e
e Prefix Utility ID with jobname . . . . . N (J - Job, S - Step, B - Both, e
e N - No) e
e Set JCL member equal to jobname . . . . N (Y - Yes, N - No) e
e Generate Job when Errors encountered . . Y (Y - Yes, N - No, W - Warnings) e
e Evaluate Multiple Exception Profiles . . A (A - All, O - One at a time) e
e Recall Migrated Spaces . . . . . N (Y - Yes, N - No) e
e Use DSNACCOR Exception Table . . . . . N (Y - Yes, N - No) e
e e
e

```

```

e HAA$JOPT ==>
E ssssssssssss Override Setup Options for DB2R8.RUN FICS ON ALL TS sssssssssssN
e HAA$JOVR ==>
e Commands: END - Return to Jobs Generation Options.
e
e
e           Current Setup values           Override values
e Job Track DB2 SubSys ==>                ==>
e Max Prim Space Alloc ==> 000050    (1-999999) C  (T,C,M) ==>
e Secondary Alloc Perc ==> 010      (1-99)% of Prim Space ==>
e Utility REGION Size ==> 0000      (0-2047)  M - (Megab) ==>          M
e DB2 Fetch Buffer Size ==> 0010      (0-2047)  M - (Megab) ==>          M
e Parallel MVS Cat LOCs ==> 10        (1-99) ==>
e Term Utility if ABEND ==> Y          (Y - Yes, N - No) ==>
e Generate STEPLIB DDs ==> Y          (Y - Yes, N - No) ==>
e Gen Copy DSNs in GMT ==> N          (Y - Yes, N - No) ==>
e
e Entering the following fields will override the calculated amount of Sort
e Work DD's space quantities and/or the number of DD's generated in the job
e Prim SortWork Space ==> 00050    (1-99999) C  (Cyls) ==>
e Second SortWork Space ==> 00050    (1-99999) C  (Cyls) ==>
e Nbr of SortWork DDs ==> 05        (1-99) ==>
e
e A blank Current Setup value indicates no value specified on Setup panel.
e

```

Utility work dataset high level . . . .	Optional
Pre-Generation User Exit Name. . . . .	Optional
Post-Generation User Exit Name . . . . .	Optional
Control Card Dataset. . . . .	
Retrieve Jobcard Dataset . .	
Member . .	
Jobname Template	(Tt,0...,#...,%...,Pppp,D...)
Override byte 1 2 3 4 5 6 7 8	

```

EsSSSSSS Utility Parns for DB2R8.RUN FICS ON ALL TS sSSSSSSSN
e HAA$JOTM
e Option ==>
e Commands: END - Return to Jobs Generation Options.
e
e Generate Templates. . . . . Y (Y - Yes, N - No)
e Generate Listdefs . . . . . Y (Y - Yes, N - No)
e
e Generate OPTION Statement:
e Preview Only . . . . . N (Y - Yes, N - No)
e Continue on Item Error . . . . N (Y - Yes, N - No)
e Return Code 0 on Warnings. . . N (Y - Yes, N - No)
e

```

*Example A-36 Job breakdown options*

```

Essssssssssssssssss Job Breakdown Options sssssssssssssssssN
e HAA$JOBR ==> e
e Commands: END - Return to Jobs Profile Update. e
e ENTER - Process new values e
e e
e Maximum nbr of jobs . . . . . 1 (0-999) e
e Maximum nbr of objects per job. . . 0 (0-99999) e
e Maximum nbr of objects per step . . 99999 (0-99999) e
e Pad Jobs if max not exceeded. . . Y (Y - Yes, N - No) e
e e
e e
e e
e e
DssssssssssssssssssM

```

Once you assign an object profile, one or more utility profiles, and, optionally, an exception profile, to a job profile, your job profile is complete. The next step is to build a job and run it or set it up so that it runs on a regular basis, for example, using a scheduler on a nightly basis. Example A-37 shows the job profile.

*Example A-37 Sample job profile screen*

```

HAA$JPRU V3R1 ----- Update Jobs Profile Display ----- 2009/10/15 20:28:06
Option ==> Scroll ==> CSR
Line Commands: A - Add D - Delete U - Update V - View
-----
Creator: DB2R8 Profile: RUN FICS ON ALL TS User: DB2R8
Share Option: U (U - Update, V - View, N - No)
Description:
Update Job Generation Options: N (Y - Yes, N - No) Row 1 of 3 >
-----
<-----
Cmd Type Order Name Creator Userid
OBJS 1 ALL OBJECTS DB2R8 DB2R8
UTIL 1 FIC DB2R8 DB2R8
EXCP 1 FIC EXCEPTION DB2R8 DB2R8
***** Bottom of Data *****

```



To build a job, go back to the Job Profile screen and use the line command “B” against that profile, as shown in Example A-38. Here the options are either to build the job online or in batch. You can also specify the name of a PDS data set and member name to house the job.

*Example A-38 Build job screen*

---

```

EsSSSSSSSSSSSSSSSS Build Job for DB2R8.RUN FICS ON ALL TS sSSSSSSSSSSSSSSSSN
e HAA$BPRD V3R1 e
e Commands: ENTER - Continue e
e END - Return to ISPF e
e ----- e
e Build Online or Batch. . 0 (0 - Online, B - Batch) e
e e e
e Edit Generated Job . . . Y (Y - Yes, N - No) e
e e e
e Build job in Dataset . . MARY.CNTL e
e Member . . BUILD e
e e e
e Job Cards: e
e ==> //DB2R8X JOB (999,POK),'MARY',CLASS=A, e
e ==> // MSGCLASS=T,NOTIFY=&SYSUID,REGION=OM e
e ==> //* e
e ==> //* e
e e e
DsSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSM

```

---

Example A-39 shows the generated JCL.

*Example A-39 The actual generated JCL*

---

```

//** * * * * *
//*
//* Step: IMC00102 *
//* *
//* Desc: This step will invoke the IBM Copy Utility *
//* *
//** * * * * *
//*
//IMC00102 EXEC PGM=DSNUTILB,REGION=0000M,COND=(4,LT),
// PARM=(DB9A,'MARYFIC')
//*
//STEPLIB DD DSN=HAA.V3R1M0.SHAALOAD,DISP=SHR
// DD DSN=FEC.V3R1M0.SFECLOAD,DISP=SHR
// DD DSN=DB9A9.SDSNEXIT,DISP=SHR
// DD DSN=DB9A9.SDSNLOAD,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//UTPRINT DD SYSOUT=*
//*
//SYSIN DD *
    TEMPLATE C1LP0001
        UNIT CART
        DSN 'DB9AU.&DB..&SN..T00025.D091015.T203137'
        RETPD 0100
        VOLCNT 5
        DISP (MOD,CATLG,CATLG)

```

```

          STACK    YES
LISTDEF CPY001U1
          INCLUDE TABLESPACE DB2R8DB.DB2R8TS1    PARTLEVEL(1)
          INCLUDE TABLESPACE DB2R8DB.DB2R8TS1    PARTLEVEL(2)
          INCLUDE TABLESPACE DB2R8DB.DB2R8TS1    PARTLEVEL(3)
          INCLUDE TABLESPACE DB2R8DB.DB2R8TS1    PARTLEVEL(4)
          INCLUDE TABLESPACE DB2R8DB.DB2R8TS1    PARTLEVEL(5)
          INCLUDE TABLESPACE DB2R8DB.DB2R8TS1    PARTLEVEL(6)
          INCLUDE TABLESPACE DB2R8DB.DB2R8TS1    PARTLEVEL(7)
          INCLUDE TABLESPACE DB2R8DB.DB2R8TS1    PARTLEVEL(8)
          INCLUDE TABLESPACE DB2R8DB.DB2R8TS1    PARTLEVEL(9)
          INCLUDE TABLESPACE DB2R8DB.DB2R8TS1    PARTLEVEL(10)
          INCLUDE TABLESPACE DB2R8DB.DB2R8TS1    PARTLEVEL(11)
          INCLUDE TABLESPACE DB2R8DB.DB2R8TS1    PARTLEVEL(12)
          INCLUDE TABLESPACE DB2R8DB.DB2R8TS1    PARTLEVEL(13)
          INCLUDE TABLESPACE DB2R8DB.DB2R8TS1    PARTLEVEL(14)
          INCLUDE TABLESPACE DB2R8DB.DB2R8TS1    PARTLEVEL(15)
          INCLUDE TABLESPACE DB2R8DB.DB2R8TS1    PARTLEVEL(16)
          INCLUDE TABLESPACE DB2R8DB.DB2R8TS1    PARTLEVEL(17)
          INCLUDE TABLESPACE DB2R8DB.DB2R8TS1    PARTLEVEL(18)
          INCLUDE TABLESPACE DB2R8DB.DB2R8TS1    PARTLEVEL(19)
          INCLUDE TABLESPACE DB2R8DB.DB2R8TS1    PARTLEVEL(20)
          INCLUDE TABLESPACE DB2R8DB.DB2R8TS1    PARTLEVEL(21)
          INCLUDE TABLESPACE DB2R8DB.DB2R8TS1    PARTLEVEL(22)
          INCLUDE TABLESPACE DB2R8DB.DB2R8TS1    PARTLEVEL(23)
          INCLUDE TABLESPACE DB2R8DB.DB2R8TS1    PARTLEVEL(24)
          INCLUDE TABLESPACE DB2R8DB.DB2R8TS1    PARTLEVEL(25)

```

```

          COPY LIST CPY001U1
CHANGELIMIT (20)
          COPYDDN      (C1LP0001)
PARALLEL      (5)
          SHRLEVEL      REFERENCE

```

```

/*
/**
/**** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
/**
/** Step:      TRM00103
/**
/** Desc:      This step will terminate the DB2 utility if the
/**              previous step has abended.  If the previous step
/**              was cancelled, the DB2 utility id will need to be
/**              terminated manually.
/**
/**** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
/**
/**
/** IF (IMC00102.ABEND) THEN
/**TRM00103 EXEC PGM=IKJEFT1A,REGION=0000M
/**STEPLIB DD DSN=HAA.V3R1M0.SHAALOAD,DISP=SHR
/**          DD DSN=FEC.V3R1M0.SFECLOAD,DISP=SHR
/**          DD DSN=DB9A9.SDSNEXIT,DISP=SHR
/**          DD DSN=DB9A9.SDSNLOAD,DISP=SHR
/**SYSTSPRT DD SYSOUT=*

```

```
//SYSTSIN DD *
  DSN SYSTEM(DB9A)
  -TERM UTILITY('MARYFIC')
END
//  ENDIF
```

---

If you build the job in batch, it generates a job that you can run regularly, which generates the utility job. Example A-40 shows this job. This is the job you would run via a scheduler.

*Example A-40 Generation of JCL for utilities*

---

```
//DB2R8X    JOB (999,POK),'MARY',CLASS=A,
// MSGCLASS=T,NOTIFY=&SYSUID,REGION=0M
//*
//*
//*
//** * * * * *
//*
//* Job Generated by IBM DB2 Automation Tool V3R1.01
//*
//* DB2 SSID: DB9A
//* SQLID:
//* Profile: DB2R8.RUN FICS ON ALL TS
//* Desc:
//* User: DB2R8
//* Date: Thursday October 15, 2009
//* Time: 20:33:49.62
//*
//** * * * * *
//*
//** * * * * *
//*
//* Step: HAA@BULD
//*
//* Desc: This job will generate the JCL for jobs profile
//*       DB2R8.RUN FICS ON ALL TS in a batch mode.
//*       The generated job will be placed in dataset
//*       MARY.CNTL(BUILD2).
//*
//* Return Codes:
//*
//* (00) - Job was built successfully with no warnings or errors
//*
//* (04) - Job was built with warning messages and the Build Job on
//*       Errors indicator was a "Y" or "W"
//*
//* (06) - Job was not built - Exception processing did not flag
//*       any objects to process.
//*
//* (08) - Job was built with error messages and the Build Job on
//*       Errors indicator was a "Y"
//*
//* (12) - Job was not built - Errors were detected and the Build Job
//*       on Errors indicator was not a "Y"
//*
```

```

/* Note: Build Job on Errors is an option in the Jobs Profile      *
/* Options screen. This option has the following values:          *
/* "Yes" - Build job on Errors or Warnings                         *
/* "No" - Do not build job on Errors or Warnings                  *
/* "Warnings" - Build job only if highest severity is a          *
/* warning message.                                              *
/*                                                                *
/*** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
/* Create temp dataset to bypass enqueue failure in ISPF          *
/*** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
//PROFILE EXEC PGM=IEFBR14
//TEMP DD DSN=%%TEMP,DISP=(NEW,PASS,DELETE),
//      UNIT=SYSDA,SPACE=(TRK,(1,1,5)),
//      DCB=ISP.SISPTENU
//*
//** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
/* Run DB2 Automation Tool Build                                  *
/*** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
//HAA@BULD EXEC PGM=IKJEFT1A,REGION=0000M
//*
//HAAERROR DD SYSOUT=*
//DLC$EXCP DD SYSOUT=*
//EXCEPTNS DD SYSOUT=*
//RUNSTATS DD SYSOUT=*
//TRIGGERS DD SYSOUT=*
//UTPRINT DD SYSOUT=*
//STPRIN01 DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//DLCDEBUG DD SYSOUT=*
//DB2PARMS DD DSN=HAA.DB9A.CONTROL,DISP=SHR
//SYSPROC DD DSN=HAA.V3R1M0.RUNTIME.CLIST,DISP=SHR
//STEPLIB DD DSN=HAA.V3R1M0.SHAALOAD,DISP=SHR
//      DD DSN=FEC.V3R1M0.SFECLOAD,DISP=SHR
//      DD DSN=DB9A9.SDSNEXIT,DISP=SHR
//      DD DSN=DB9A9.SDSNLOAD,DISP=SHR
//ISPLLIB DD DSN=HAA.V3R1M0.SHAALOAD,DISP=SHR
//      DD DSN=FEC.V3R1M0.SFECLOAD,DISP=SHR
//      DD DSN=DB9A9.SDSNEXIT,DISP=SHR
//      DD DSN=DB9A9.SDSNLOAD,DISP=SHR
//ISPPLIB DD DSN=HAA.V3R1M0.SHAAPENU,DISP=SHR
//      DD DSN=FEC.V3R1M0.SFECPENU,DISP=SHR
//ISPTLIB DD DSN=%%TEMP,DISP=(OLD,DELETE,DELETE)
//      DD DSN=ISP.SISPTENU,DISP=SHR
//ISPMLIB DD DSN=ISP.SISPMENU,DISP=SHR
//      DD DSN=HAA.V3R1M0.SHAAMENU,DISP=SHR
//      DD DSN=FEC.V3R1M0.SFECMENU,DISP=SHR
//ISPPLIB DD DSN=HAA.V3R1M0.SHAASLIB,DISP=SHR
//ISPPROF DD DSN=%%PROF,DISP=(NEW,DELETE),
//      UNIT=SYSDA,SPACE=(TRK,(2,1,2)),
//      DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
//ISPFIL DD DSN=MARY.CNTL,DISP=SHR
//ISPLLOG DD SYSOUT=*,DCB=(RECFM=VA,LRECL=125,BLKSIZE=129)
//ISPWRK1 DD UNIT=SYSDA,SPACE=(CYL,(30,30)),
//      DCB=(RECFM=FB,LRECL=133,BLKSIZE=1330)

```

```

//SYSTSIN DD *
  PROFILE NOPREFIX
  ISPSTART PGM(HAA@BULD)
//*
//HAA#DATA DD *
  GENERATE UTILITY JOB (
    DB2_SUBSYSTEM DB9A
    USER_INDICATOR HAA
    PROFILE_NAME 'RUN FICS ON ALL TS'
    PROFILE_CREATOR DB2R8
    PROFILE_DESCRIPTION ''
    EXECUTION_LIB_2 HAA.V3R1M0.SHAALOAD
EXECUTION_LIB_4 FEC.V3R1M0.SFECLOAD
    GEN_TO_DATASET MARY.CNTL
    DEBUG_MODE OFF
    GEN_TO_MEMBER BUILD2
    JOB_CARD_1_1 '//DB2R8X JOB (999,P0K),"MARY",CLASS=A'
    JOB_CARD_1_2 ','
    JOB_CARD_2_1 '// MSGCLASS=T,NOTIFY=&SYSUID,REGION=0M'
    JOB_CARD_3_1 '//*'
    JOB_CARD_4_1 '//*'
  )
//*

```

---

As you can see, the DB2 Automation Tool can help you generate utility JCL easily and repeatedly. You can set up standard utility profiles and standardize the utility execution by application. It is a powerful tool that can help you become very productive by allowing the tool to generate these utilities on a regular basis and even on an as needed basis.

## A.4 DB2 High Performance Unload for z/OS

The DB2 High Performance Unload for z/OS (HPU) product is intended to unload table spaces using superior performance techniques that minimize CPU and elapsed time. It works outside DB2, and accesses the DB2 linear VSAM data sets directly without using the DB2 buffer pools. This direct access takes maximum advantage of the VSAM buffering capability by allowing an entire cylinder to be read in a single I/O. HPU allows you to run several unload jobs that access the same table space in parallel. You also have an option to run inside of DB2 when you are using SQL that cannot be performed against the data outside of DB2's control.

This tool was developed before DB2 support was in place for the UNLOAD utility. When compared to the UNLOAD utility and its performance characteristics, in general, HPU uses similar elapsed time, but usually less CPU than the DB2 UNLOAD utility.

Some of the high-level features of HPU are:

- ▶ The capability to process several unloads that access the same table space in parallel
- ▶ Unload against an image copy
- ▶ Unload selected rows and columns from a table space
- ▶ Supports full SQL SELECT capability
- ▶ Unload every 'n' rows (sampling) or unload a certain number of rows
- ▶ Generates load control statements that can be used for load

- ▶ Can unload to different output files
- ▶ User exit available to modify, discard, or view rows
- ▶ Can use multiple formats during the unload process:
  - DSNTIAUL compatible
  - VARIABLE
  - COMMA DELIMITED
  - USER DEFINED
    - User defined allows you to specify different conversion algorithms
  - EXTERNAL format

HPU can also be integrated with the DB2 Administration Tool. As a result, you can optionally generate HPU syntax for migrate (MIG command), ALTER TABLESPACE (ALC command), and utility execution (UTIL command).

For more information about DB2 High Performance Unload for z/OS, refer to the following address:

<http://www.ibm.com/software/data/db2imstools/db2tools/db2hpu-zos/>

## A.5 DB2 Recovery Expert for z/OS

From a backup and recovery perspective, the challenge is to be prepared and recover quickly to minimize application outage. Our recommendation is to ensure you have proper backup and recovery strategies in place. Recovery is not something you do every day and when you are under a time constraint, to get objects recovered quickly, time is of the essence. DB2 Recovery Expert for z/OS is a tool to minimize the pressure DBAs feel during a recovery scenario, when there is much pressure to ensure that the object(s) are available for the application as quickly as possible. The Recovery Expert Tool analyzes the difference scenarios for recovery and is less error prone than you trying to manage the recovery and the different choices by yourself, especially when there is added pressure from management and the user to get the application up and running quickly.

The Recovery Expert Tool comes with two distinct interfaces: ISPF and a graphical user interface (GUI). Each one is used for different purposes.

### A.5.1 Recovery Expert features

Recovery Expert simplifies the recovery process for you by providing a GUI interface. You provide information about what needs to be recovered and Recovery Expert reviews the different assets available, such as full image copies, incremental image copies, and copies made with DSN1COPY. Recovery Expert provides available recovery options and gives you relative costs of the different strategies. For example, for a point-in-time recovery, Recovery Expert might suggest you follow one of the following strategies:

- ▶ Back out the changes by applying the most recent image copies, and have DB2 perform LOGAPPLY for those changes recorded in the DB2 logs.
- ▶ Apply the most recent image copy and run REDO SQL generated from the DB2 log by Recovery Expert.
- ▶ Run UNDO SQL generated from the DB2 log by Recovery Expert to the current object.

- ▶ Use a system level backup as a recovery base for the table space and apply log changes performed since the BACKUP SYSTEM was taken.
- ▶ Perform a RECOVER LOGONLY after a DSNCOPY.

In each of the possible recovery scenarios, based on your recovery base assets, the Recovery Expert tool provides a relative cost estimate for each of these different plans. You usually want to choose the quickest and most efficient plan to recover. However, you still need to make the decision as to which strategy to choose to perform the recovery. Because you know the application best, you can make a better decision by choosing the fastest and least intrusive approach to performing the recovery.

Recovery Expert also provides a mechanism to restore a dropped object and discover quiet times on the log for an object or a set of objects. Quiet time is where there are no UOWs against these objects and a RBA or LRSN value is provided in case you need to consider a recovery to a point in time.

## A.5.2 BACKUP SYSTEM and RESTORE SYSTEM

FlashCopy technology is gaining more interest for and wide acceptance by customers struggling with providing high availability in a 24x7 environment. Many sites are experiencing shorter batch windows and are looking toward more storage-aware tools to bridge that gap between the complicated technology and the fast backup strategy this technology affords. We describe the use of these utilities in Chapter 13, “BACKUP and RESTORE SYSTEM utilities” on page 339.

The Recovery Expert product provides valuable assistance to the BACKUP SYSTEM utility by performing validity checking via an ISPF interface. One of its main features ensures that the BACKUP SYSTEM process produces a valid backup that can be used for a SYSTEM RESTORE. You do not want to rely on a backup and learn that when you are trying to do a restore after a critical problem occurred that your backup cannot be used in a RESTORE SYSTEM.

The type of checking Recovery Expert uses is extensive and includes, but is not limited to, the following validations:

- ▶ Ensures the z/OS user catalogs are included in the backup.
- ▶ Ensures that all source and target volumes are correct, are the same device type, and are online.
- ▶ Ensures that all volumes are included in the backup.
- ▶ Ensures the target volumes are not in use by another backup profile.
- ▶ Ensures that DB2 log and object data are on separate volumes.
- ▶ Ensures that DB2 log data and object data are cataloged in separate z/OS user catalogs. If they are not, you can still run a SYSTEM BACKUP, but you can only restore the full subsystem.
- ▶ Ensures that all DB2 logs are included in a FULL system backup.

Recovery Expert shows, via an ISPF interface, the DASD usage of the DB2 subsystem and what may need to be corrected before effectively implementing system level backup and restore methods. In addition, it can generate JCL to move object data to appropriate volumes in order to segregate DB2 log data from object data. It also provides support for moving and renaming data sets that contain DB2 logs, BSDS, and object data, as examples.

When you need to restore data using SYSTEM RESTORE, it provides a strategy to restore a set of objects or the whole system. It can also automatically recover or rebuild objects placed in RECP/RBDP after the RESTORE SYSTEM process. When the SYSTEM RESTORE completes, it not only reports on objects that were not restored properly, but it can optionally produce the necessary JCL to recover or rebuild those objects.

In addition, Recovery Expert handles a number of BACKUP SYSTEM and RESTORE SYSTEM limitations. Here are a few examples:

- During a recovery situation, you have a system level backup, but since that backup was taken, a REORG TABLESPACE occurred. Ordinarily, DB2 would use the system level backup as a recovery point for this pageset, but since a reorganization of the table space occurred, DB2 cannot do the LOGAPPLY for this object and it places it in a recover pending status (RECP).

Recovery Expert can help here: It drives the RESTORE SYSTEM and detects when objects are placed in RECP. It then generates the JCL to recover these objects from the last image copy taken. The benefit is that you do not have to manually find these objects and create the JCL; Recovery Expert does this process for you. All you need to do is submit the jobs generated by the tool.

- If you are using RESTORE SYSTEM for data set recovery, and the data set moved from its original location since the BACKUP SYSTEM ran, then you cannot use recover for this object. The reason for this is that DFSMSHsm simply does not know where it resided when the backup was taken.

On the other hand, if the system level backup was taken by Recovery Expert, then the tool knows exactly where the source was when that backup was taken. This restriction is lifted in z/OS V1.11. However, because DB2 has to put the object back on its original volume, and if there is insufficient space on that volume for the table space, the recovery will fail. Recovery Expert handles this situation and can place the data sets anywhere within SMS pools, even if this is a multi-volume data set.

You can see that Recovery Expert really provides value not only in these recovery situations but also in the complicated setup to facilitate a recovery from a system level backup. For more information about the BACKUP SYSTEM and RESTORE SYSTEM utilities, refer to Chapter 13, “BACKUP and RESTORE SYSTEM utilities” on page 339.

Recovery Expert supports multiple different vendors using the Flash Copy type of technology. You do not have to be a storage management expert to use this product. For more information about the Recovery Expert product, refer to:

<http://www.ibm.com/software/data/db2imstools/db2tools/db2re-zos/>

In addition, refer to *IBM DB2 9 for z/OS: New Tools for Query Optimization*, SG24-7421.



## A.6 DB2 Change Accumulation Tool

The DB2 Change Accumulation Tool creates full image copies of objects without the need to start or stop the database. By using this low-impact tool, you can minimize database downtime and reduce recovery time.

The space you want to make a copy of must have a complete, valid full image copy registered in the SYSCOPY catalog table. The DB2 Change Accumulation Tool takes the full image copy, applies any incremental image copies, and then applies log data up to the specified recovery point. The resulting file is logged in SYSIBM.SYSCOPY as a primary local or backup full SHRLEVEL REFERENCE image copy that can be used to recover the object.

This process is similar to MERGECOPY, except that the DB2 Change Accumulation Tool allows you to select specific points up to which you want the image copy created. For example, you can select the TOCURRENT recovery point, or a specific RBA or LRSN value from the DB2 log.

The DB2 Change Accumulation Tool also allows you to create a mini-log of the changes made to an object. The mini-log is essentially all of the inserts, updates, and deletions made to the object since the last image copy.

Once you have a mini-log created, in order to recover an object, there is a WRITE-TO-VSAM option that allows the Change Accumulation Tool to update the underlying VSAM data set and apply the changes from the mini-log data set for faster recovery. The mini-log can also be applied toward an image copy, producing another updated SHRLEVEL REFERENCE image copy. This process can be accomplished with taking an outage to the underlying object.

The Change Accumulation Tool uses object, utility, and job profiles in a similar way as the DB2 Automation Tool. Refer to “*DB2 Automation Tool*” on page 435 for more information about these profiles.

For more information about the DB2 Change Accumulation Tool, go to the following address:

<http://www.ibm.com/software/data/db2imstools/db2tools/db2-change-accumulation/>

## A.7 DB2 Storage Management Utility

The DB2 Storage Management Utility (SMU) is a generalized tool that allows for different functions, especially related to the management of DB2 data set allocation and usage. With this tool, you can resize objects and generate utilities to manage the reallocation. In addition, there is the capability to generate utilities based on user-defined criteria. Unfortunately, SMU does not use RTS.

For more information about SMU, go to the following address:

<http://www.ibm.com/software/data/db2imstools/db2tools/db2smu-zos/>

## A.8 Optim Query Tuner and Optim Query Workload Tuner

Collecting statistics on all columns of a table with RUNSTATS TABLE(ALL) can be very costly and completely unnecessary depending on the application. The recommendation for RUNSTATS is to only collect statistics on those columns that are necessary, presumably on those columns that are used as search conditions in the WHERE clause for an application or as join columns. However, your application may have several hundred SQL statements, and to examine each WHERE clause can be a daunting task.

Having a tool such as a Statistics Advisor can be very helpful in determining the best possible RUNSTATS statement that should be used based on the SQL in an application. Not all SQL is static; you can certainly collect the SQL for a static application using the DB2 catalog, but what do you do for dynamic SQL? The Statistics Advisor is an integral component of both the Optim™ Query Tuner and the Optim Query Workload Tuner products, and is used for collecting SQL for dynamic SQL.

An example of the recommendations provided by the tools for statistics collection is shown in Figure A-1.

### Improve Statistics Quality and Collection

➡ Provides advice on

- Missing statistics
- Conflicting statistics
- Out-of-date statistics

➡ Results

- Accurate estimated costs
- Better query performance
- Less CPU consumption
- Improved maintenance window throughput

#### Statistics Recommendation Detail

**Generates RUNSTATS control statements**

RUNSTATS INDEX(SYSADM.PXS@SKQK FREQVAL NUMCOLS 1 COUNT 15)  
SHRLEVEL CHANGE REPORT YES UPDATE ALL HISTORY NONE

RUNSTATS TABLESPACE DB4LINE1.TSLINE1  
TABLE(SYSADM.LINEITEM) SAMPLE 40  
COLUMN(L\_SHIPMODE)  
COLGROUP(L\_SHIPDATE) HISTOGRAM NUMQUANTILES 25  
COLGROUP(L\_RECEIPTDATE) FREQVAL COUNT 15 HISTOGRAM NUMQUANTILES 25  
COLGROUP(L\_SHIPMODE) FREQVAL COUNT 15  
COLGROUP(L\_RETURNFLAG) FREQVAL COUNT 15  
SORTDEVT SYSQA SORTNUM 4  
INDEX(SYSADM.SXL@PKSKOKEPDSQN KEYCARD,

**Indicates conflicting and missing statistics**

**Conflicting statistics explanation**

**Conflicts detail**

TABLE SYSADM.LINEITEM  
One of the frequency records (-1.0) of the L\_ORDERKEY column group is out of range [0,1]  
Tolerance: 0.0010

The maximum frequency of the column group or column (L\_ORDERKEY), (0.0), is less than the average frequency, or 1 divided by the greater than the average unless only least-frequently occurring values are being collected.  
Tolerance: 0.0010

Figure A-1 Sample of Statistics Advisor usage

Optim Query Tuner supports both DB2 for LUW and DB2 for z/OS. It allows you to optimize query performance and is generally more suitable for application developers working in a development environment, as it tunes a single query. The Optim Query Workload Tuner is a product that is more suited for production; it has the capability to analyze multiple queries at once. Both of these products have a component that gives you recommendations for the statistics to collect using the RUNSTATS utility as a complete RUNSTATS control statement.

For dynamic statements, the Optim Query Workload tuner allows you to define and select a workload that consists of a number of queries. It can get the workload from the dynamic statement cache where the specific number of occurrences of each query is kept. You can also schedule periodic snapshots of the dynamic statement cache.

The tool knows the number of times each query ran, and captures the average CPU time and execution counts. This is important because query A could use 0.1 seconds of CPU time, but only ran 5 times in a 24 hour period, and query B may use 0.05 seconds of CPU time, but ran 100 K times. Obviously, query B is the one on which you may want to focus your attention. Both tools allow for drill-down into the details of each SQL statement, where you can review join predicates, where clauses, object statistics, and much more additional information. It can inform you if the data is skewed, and allow you to see both the original query and what was transformed by DB2 in the case where DB2, for example, might create a virtual table for a correlated subquery.

It shows you a visual access path similar to what Visual Explain displayed, and provides advice about missing statistics, conflicting statistics, and out of date statistics. The most important feature from a utilities perspective is that each tool generates the RUNSTATS control statement.

You can import statistics from the DB2 catalog. You have the ability to filter out those queries you are not interested in and manipulate the workload as though you were running it in production, by, for example, using a higher number of occurrences.

The Statistics Advisor component is an integral part of the following set of products:

- ▶ Visual Explain
- ▶ DB2 Optimization Service Center (OSC) V1.1 FP6
- ▶ DB2 Optimization Expert for z/OS (OE) V2.1
- ▶ Optim Query Tuner for z/OS (OQT) V2.2
- ▶ Optim Query Workload Tuner for z/OS (OWQT) V2.2

In addition, if you want to analyze a collection of SQL, namely a workload, then the Workload Statistics Advisor component is what you need. The Workload Statistics Advisor is part of the following set of products:

- ▶ DB2 Optimization Service Center V1.1 FP6
- ▶ DB2 Optimization Expert for z/OS V2.1
- ▶ Optim Query Workload Tuner for z/OS V2.2

**Note:** The product Visual Explain is supported until end of service for DB2 V8.

Besides the Statistics Advisor and the Workload Statistics Advisor, there are additional components to the various tools they are a component of, and these components, as well as the different tools, are listed in Table A-2 on page 462.

**Note:** The current Optimization Service Center remains supported through DB2 9. All its functions are currently included in Optim Query Tuner for z/OS and Optim Query Workload Tuner for z/OS.

The Optimization Service Center (OSC) is a part of DB2 and is a no-fee based product. It includes the Query Statistics Advisor. In July 2009, Optimization Service Center was renamed Optim Query Tuner for z/OS.

Optimization Expert (OE) is a fee-based product and includes the Query Statistics Advisor and the Workload Query Statistics Advisor. In July 2009, Optimization Expert was renamed Optim Query Workload Tuner for z/OS. The different functions of each product are summarized in Table A-2, including the Statistics Advisor and the Workload Statistics Advisor.

*Table A-2 Different functions in OSC, OE, OQT, and OWQT*

<b>Component</b>	<b>Optimization Service Center V1.1 FP6</b>	<b>Optimization Expert V2.1</b>	<b>Optim Query Tuner for z/OS V2.2</b>	<b>Optim Query Workload Tuner for z/OS V2.2</b>
Queries from all sources	Yes	Yes	Yes	Yes
Query Formatter	Yes	Yes	Yes	Yes
Query Annotation	Yes	Yes	Yes	Yes
Access Plan Graph	Yes	Yes	Yes	Yes
Visual Plan Hint	Yes	Yes	Yes	Yes
Query Statistics Advisor	Yes	Yes	Yes	Yes
Workload Statistics Advisor	Yes	Yes	No	Yes
Profile based Monitoring	Yes	Yes	No	Yes
Query Index Advisor	No	Yes	Yes	Yes
Workload Index Advisor	No	Yes	No	Yes
Query Advisor	No	Yes	Yes	Yes
Query Workload Advisor	No	Yes	No	Yes
Access Path Advisor	Yes	Yes	Yes	Yes
Workload Service SQL	Yes	Yes	No	Yes
Query Env Cap (SvcSQL)	Limited	Limited	Limited	Limited

For more information about the these products, refer to the addresses shown in Table A-3.

Table A-3 Optim products and links

Product	Link
Optim Query Tuner for z/OS V2.2	<a href="http://www.ibm.com/software/data/optim/query-tuner-z/">http://www.ibm.com/software/data/optim/query-tuner-z/</a>
Optim Query Workload Tuner for z/OS V2.2	<a href="http://www.ibm.com/software/data/optim/query-workload-tuner-z/">http://www.ibm.com/software/data/optim/query-workload-tuner-z/</a>

## A.9 DB2 Buffer Pool Analyzer for z/OS

The DB2 utilities use the DB2 buffer pools, so be sure that your DB2 buffer pools are optimally sized and thresholds are set accordingly to provide the utilities with the best possible performance. The DB2 Buffer Pool Analyzer for z/OS (BPA) product is a stand-alone product and is also a component of the Tivoli OMEGAMON® XE for DB2 Performance Expert on z/OS product. There is a graphical user interface (GUI) that is used to generate different reports and do the analysis.

The BPA contains the following functionality:

- ▶ Collects buffer pool data trace data.
  - Either as summary or detailed data
  - Either continuously or in sampling mode
  - Collection runs in TSO session or batch
- ▶ Generates various reports and displays results in multiple formats. Different reports are generated in TSO and in the GUI.
- ▶ Provides expert knowledge and recommendations.
- ▶ Recommends object placements, BP size, and thresholds.
- ▶ Generates ALTER statements for the recommendations.
- ▶ Provides simulation for planned changes so you can verify the expected results.

Effectively, the BPA provides different mechanisms to get the most out of your buffer pools by sizing the buffer pools adequately and improving resource utilization. A typical tuning approach is to conduct a summary analysis first and help you determine the access characteristics of the objects in the pool. Plan to separate the predominantly random objects from those that are predominantly sequential. You probably will not need very large pools for sequential objects. You will, however, need a large pool for the random objects to increase the page residency time.

For more information about the DB2 Buffer Pool Analyzer for z/OS product, go to the following address;

<http://www.ibm.com/software/data/db2imstools/db2tools/bpa/index.html>





# Utilities related maintenance

In this appendix, we look at recent maintenance for DB2 9 for z/OS that generally relates to performance and availability for utilities.

These lists of APARs represents a snapshot of the current maintenance for performance and availability functions at the moment of writing. As such, the list will become incomplete or even incorrect at the time of reading. It is here to identify areas of performance or functional improvements.

Table B-1 shows a summary of utilities related maintenance.

*Table B-1 List of relevant maintenance for DB2 9 for z/OS*

APAR #	Area	Text	PTF
II13495	DFSORT	How DFSORT takes advantage of a 64-bit real architecture.	N/A
II14047	DFSORT	Information APAR about DB2 Version 8 Utilities' exclusive use of DFSORT.	N/A
II14213	DFSORT	Continuation of II14047.	N/A
PK41182	REORG	Displays claimers about drain failure.	UK24255 also V8
PK41711	CHECK	Runs data consistency checks without impacting BACKUP SYSTEM or disk mirroring.	UK41370 also V8
PK41899	DFSORT	Greater utility parallelism with SORTNUM elimination.	UK33636
PK42573	COPY	CHANGELIMIT(ANY) creates a full image copy if any page has been updated.	UK25468
PK42768	COPY	Companion APAR of PK42573.	UK25469
PK47083	LOAD and REORG	Auto-invalidate of cached dynamic statements on completion.	UK34198

APAR #	Area	Text	PTF
PK51853	LOAD and REORG	Removes the limit of 254 partitions of a table space with the COMPRESS YES attribute.	UK31489 also V8
PK55972			UK39612
PK60612	UNLOAD	Allows UNLOAD from an ICOPY of a table space that was non-segmented, even though now it is defined as segment.	UK35132
PK60956	SORTBLD	This is a change in force write logic to allow performance improvement in SORTBLD for indexes with small SECQTY.	UK34808 also V8
PK61759	LOAD & REORG	Improving performance for LOAD, REORG, and any utilities that invoke DFSORT.	UK36306 also V8
PK63324	LOAD	LOAD COPYDICTIONARY option.	UK37144
PK63325	LOAD	LOAD COPYDICTIONARY option.	UK37143
PK67154	REORG	REORG PART of empty partition now avoids NPI scan for non-clustering indexes.	UK45138 also V8
PK70269	TEMPLATE	Dynamically allocates a z/OS UNIX file (HFS) for the LOAD and UNLOAD utilities.	UK43948 also V8
PK70866	CREATE	Retrofit V9 preformat.	UK42884 V8 only
PK74993	COPY	20% elapsed time improvement for copy of multiple small data sets to tape.	UK45791 also V8
PK75216	LOAD & UNLOAD	LOB file reference variable performance (PDS).	UK43355
PK75643	DSNZPARM OPTIOWGT	Default changed to enable improved formula for balancing the costs of input/output and CPU speeds.	UK42565
PK76860	Cross loader	Performance improvement for CCSID data conversion.	UK46236 also V8
PK77313	UNLOAD	Performance for multi-table table spaces. UTILINIT phase uses DBD rather than catalog lookup.	UK43512 also V8
PK77426	DSNZPARM OPTIXOPREF	Default changed to enable favoring an index-only index over an index-plus-data index when the index-only index has equal or better filtering.	UK51283
PK78516	DSN1COPY	Improvement for DSN1COPY I/O performance.	UK43977
PK78558	All Utilities	Write diagnostic log record at utility termination so IFCID 306 readers can trigger refresh.	N/A
PK78865	COPY	Improvement performance when processing very large LISTDEF lists.	UK45998
PK78958	REORG	Disables reordered row format for compressed table space.	UK45353



APAR #	Area	Text	PTF
PK79136 PE is PK92248	SORT	Reduction of storage consumption of DFSORT in parallel invocations.	UK44703
PK81232	COPY	Improves performance of copy partitioned table spaces.	UK45192
PK83096	COPY	Improves performance of copy LOB shrlevel change in data sharing.	UK47616
PK83996	SYSUTILX	Space reuse problem in catalog/directory with LOBs and scrollable cursors in data sharing.	UK46749 also V8
PK85856	SORT	zIIP enablement.	UK48846
PK85881	LOAD/ REOG	ROWFORMAT support enablement.	UK50413
PK85889	SORT	DB2 enablement APAR for zIIP offload in DFSORT.	UK48912
PK87348	Reordered Row Format	New DSNZPARM SPRMRRF= ENABLE/DISABLE and LOAD REPLACE and REORG ROWFORMAT conversion support.	UK50412
PK87762	REORG	Reorg for non-consecutive partitions in a single statement.	OPEN
PK89645	BACKUP SYSTEM	DATA COMPLETE LRSN message and accuracy.	UK51104 also V8
PK89889	DFSORT	Abend U1094.	UK50430
PK92248	REORG and LOAD REPLACE	Overrides large TMAXLIM installation options in DFSORT.	UK49339
PK92725	BACKUP and RESTORE	Removes RECOVER restrictions.	OPEN
PK97713	DFSORT	Failed DFSORT license check after PK85889.	UK51396 V8 only
PM03691	DSN1COPY	Functions to help with mismatching object definitions.	OPEN



# Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

## IBM Redbooks

For information about ordering these publications, see “How to get Redbooks” on page 470. Note that some of the documents referenced here might be available in softcopy only.

- ▶ *DB2 9 for z/OS: Backup and Recovery I/O Related Performance Considerations*, REDP-4452
- ▶ *DB2 9 for z/OS: Buffer Pool Monitoring and Tuning*, REDP-4604
- ▶ *DB2 9 for z/OS Performance Topics*, SG24-7473
- ▶ *DB2 9 for z/OS Technical Overview*, SG24-7330
- ▶ *DB2 for MVS/ESA Version 4 Non-Data-Sharing Performance Topics*, SG24-4562
- ▶ *DB2 for OS/390 Version 5 Performance Topics*, SG24-2213
- ▶ *DB2 UDB for OS/390 Version 6 Performance Topics*, SG24-5351
- ▶ *DB2 UDB for z/OS: Design Guidelines for High Performance and Availability*, SG23-7134
- ▶ *DB2 UDB for z/OS Version 8: Everything You Ever Wanted to Know, ... and More*, SG24-6079
- ▶ *DB2 UDB for z/OS Version 8 Performance Topics*, SG24-6465
- ▶ *DB2 for z/OS and OS/390 Version 7 Performance Topics*, SG24-6129
- ▶ *DB2 for z/OS and OS/390 Version 7 Using the Utilities Suite*, SG24-6289
- ▶ *Disaster Recovery with DB2 UDB for z/OS*, SG24-6370
- ▶ *Enterprise Data Warehousing with DB2 9 for z/OS*, SG24-7637
- ▶ *IBM DB2 9 for z/OS: New Tools for Query Optimization*, SG24-7421
- ▶ *IBM System p5 520 and 520Q Technical Overview and Introduction*, REDP-4137
- ▶ *LOBs with DB2 for z/OS: Stronger and Faster*, SG24-7270
- ▶ *Optimizing Restore and Recovery Solutions with DB2 Recovery Expert for z/OS V2.1*, SG24-7606

## Other publications

These publications are also relevant as further information sources:

- ▶ *DB2 Version 9.1 for z/OS Administration Guide*, SC18-9840
- ▶ *DB2 Version 9.1 for z/OS Application Programming Guide and Reference for Java*, SC18-9842
- ▶ *DB2 Version 9.1 for z/OS Application Programming and SQL Guide*, SC18-9841
- ▶ *DB2 Version 9.1 for z/OS Codes*, GC18-9843

- ▶ *DB2 Version 9.1 for z/OS Command Reference*, SC18-9844
- ▶ *DB2 Version 9.1 for z/OS Data Sharing: Planning and Administration*, SC18-9845
- ▶ *DB2 Version 9.1 for z/OS Installation Guide*, GC18-9846
- ▶ *DB2 Version 9.1 for z/OS Internationalization Guide*, SC19-1161
- ▶ *DB2 Version 9.1 for z/OS Introduction to DB2 for z/OS*, SC18-9847
- ▶ *DB2 Version 9.1 for z/OS Messages*, GC18-9849
- ▶ *DB2 Version 9.1 for z/OS ODBC Guide and Reference*, SC18-9850
- ▶ *DB2 Version 9.1 for z/OS Performance Monitoring and Tuning Guide*, SC18-9851
- ▶ *DB2 Version 9.1 for z/OS RACF Access Control Module Guide*, SC18-9852
- ▶ *DB2 Version 9.1 for z/OS Reference for Remote DRDA Requesters and Servers*, SC18-9853
- ▶ *DB2 Version 9.1 for z/OS SQL Reference*, SC18-9854
- ▶ *DB2 Version 9.1 for z/OS Utility Guide and Reference*, SC18-9855
- ▶ *DB2 Version 9.1 for z/OS What's New?*, GC18-9856
- ▶ *DB2 Version 9.1 for z/OS XML Guide*, SC18-9858
- ▶ *DFSMSHsm Fast Replication Technical Guide*, SG24-7069
- ▶ *DFSORT Application Programming Guide*, SC26-7523
- ▶ *IBM OmniFind Text Search Server for DB2 for z/OS Installation, Administration, and Reference Version 1 Release 1*, GC19-1146
- ▶ *IBM Spatial Support for DB2 for z/OS User's Guide and Reference Version 1 Release 2*, GC19-1145
- ▶ *IRLM Messages and Codes for IMS and DB2 for z/OS*, GC19-2666
- ▶ *z/OS V1R10.0 DFSMS Storage Administration Reference (for DFSMSdftp, DFSMSdss, DFSMSHsm)*, SC26-7402
- ▶ *z/OS V1R11.0 TSO/E Command Reference*, SA22-7782

## Online resources

These Web sites are also relevant as further information sources:

- ▶ DB2 for z/OS support  
<http://www.ibm.com/software/data/db2/support/db2zos/>

## How to get Redbooks

You can search for, view, or download Redbooks, Redpapers, Technotes, draft publications and Additional materials, as well as order hardcopy Redbooks publications, at this Web site:

[ibm.com/redbooks](http://ibm.com/redbooks)

## Help from IBM

IBM Support and downloads

[ibm.com/support](https://ibm.com/support)

IBM Global Services

[ibm.com/services](https://ibm.com/services)



# Index

## Symbols

/II14213 20

## Numerics

37 91, 173, 309, 356, 442  
4096 partitions 28  
5655-N97 19  
93 active log data sets 12

## A

access control 183  
ACCESSPATH 296, 303, 404  
ALTER BUFFERPOOL 394  
ALTER TABLE 31, 37, 127, 214, 260, 301, 335  
application 5, 28, 45, 76, 90, 116, 191–192, 212, 236, 253, 324, 342, 375, 377, 455–456  
ASCII 178  
ATAWKnn data sets 144  
attribute 9, 39, 77, 177, 185, 201, 225, 253, 323, 355  
auxiliary index 190, 217, 330  
auxiliary table 6, 316, 321  
AUXW 6, 331  
AVGROWLEN 104, 160, 446

## B

BACKUP xxiii, 5, 225, 252, 340, 342, 404, 457  
BACKUP SYSTEM 5, 252, 340, 404, 457  
base table 6, 11, 78–79, 125, 168, 190, 206, 258, 302, 316, 321  
BIGINT 4, 188, 331  
BIT 20, 141  
BLOB 190, 192, 335  
BOTH 295, 305  
broken page 232–233  
BSDS 12, 270, 342, 398, 457  
Buffer Pool Analyzer 140  
buffer pools 140, 281, 375, 393, 455, 463  
BUILD2 18, 22, 31, 76, 86, 215, 453, 455

## C

CACHE 403  
CARDF 97, 294, 446–447  
CATENFM 6  
CATMAINT 6, 55, 214, 390  
CCSID 13, 171–172, 179, 430  
1208 179  
CF 394  
Change Accumulation Tool 140  
CHANGELIMIT 103, 231, 418, 452  
CHECK 6, 31, 50, 89, 232, 258, 299, 321, 370, 390, 429, 435  
check constraints 6, 199, 321

CHECK DATA 6, 55, 77, 139, 141, 159, 321–322, 380, 435  
CHECK INDEX 6, 67, 89, 141, 159, 225, 321, 323, 380, 405  
CHECK LOB 6, 70, 141, 159, 321, 380  
check page 232  
CHECKPAGE 16, 418  
chunk 218  
chunking 218  
CICS xxiv, 407  
class 16, 147, 193, 213, 227, 252, 326, 370, 377, 384, 442  
CLOB 182, 190  
clone table 14, 168, 235, 263, 311, 335, 418  
CLUSTER 29, 37, 243, 335, 380  
clustering 29  
COEXIST 393  
COLGROUP 144, 291, 295  
components 461  
compression 13–14, 36, 94, 115, 175–176, 212, 276, 292, 299, 397, 421  
compression dictionary 17, 23, 135, 179, 299, 397  
Concurrent Copy 16, 240  
condition 33, 147, 188, 433  
CONNECT 196, 407–408  
CONTINUE 60, 213, 383  
COPY 7, 35, 50, 60–61, 76, 95, 103, 170, 173, 184, 207–208, 223–224, 253, 323, 344, 383, 390, 393, 416–417, 466–467  
CHECKPAGE 232–233  
copying to disk 236  
copying to tape 236  
FULL 56, 229, 272, 356, 418, 439  
INCREMENTAL 235, 242  
lists 72, 227  
parallelism 80, 228  
SHRLEVEL CHANGE 80, 126, 224, 236, 262, 324  
SHRLEVEL REFERENCE 80, 140, 229, 236, 271  
COPY PARALLEL 50, 80, 228  
Copy Pool 5, 21, 252  
COPY YES 9, 53, 77, 123, 224, 226, 277, 330, 355  
COPYDDN 56, 61, 77, 120, 170, 172, 185, 207–208, 227, 234, 277, 396, 430–431  
copying indexes 224  
recommendations 226  
COPYP 77, 184  
COPYPOOL 341, 346  
COPYTOCOPY 7, 23, 50, 60, 223, 237, 390, 439  
FROMCOPY 239  
FROMLASTCOPY 239  
FROMLASTFULLCOPY 239  
FROMLASTINCRCOPY 239  
COPYTOCOPY OPTIONS 239  
core utilities 19  
COUNT 230, 273, 295, 305, 383

CREATE TABLE  
 LIKE 127  
 CREATOR 118, 129, 196, 381  
 Cross Loader 17  
 cross loader 195–196  
 CS 116, 199, 396–397  
 CTHREAD 90, 263, 386–387  
 cursor stability 199

## D

DAnnWKmm data sets 144  
 data propagation 14  
 data set extents 112, 114  
 data set sizing 55, 60, 143  
 data sharing xxiv, 5, 29, 99–100, 247, 252, 347, 388, 467  
 database name 382  
 Database System Server 193  
 DATACLAS 325, 378  
 data-partitioned secondary indexes 31  
 DATASIZE 122  
 DATE 56, 220, 239, 307, 313, 346  
 DB2 Administration Tool 111, 139, 414  
 DB2 Administrative Task Scheduler 136  
 DB2 Automation Tool 112, 139, 302, 395, 429, 435  
 DB2 Change Accumulation Tool 288, 459  
 DB2 High Performance Unload 139, 455  
 DB2 High Performance Unload for z/OS 455  
 DB2 Optimization Expert for z/OS 461  
 DB2 Optimization Service Center 461  
 DB2 Recovery Expert for z/OS 456  
 DB2 Storage Management Utility 459  
 DB2 Utilities Suite 19–20, 142, 169  
 DB2 Utility Enhancement Tool 139  
 DB2 Utility Enhancement Tool for z/OS 429  
 DBCLOB 182, 190  
 DBCS 182  
 DBD 8, 40, 175, 281, 337, 396  
 DBET 35, 37, 384, 389  
 DBNAME 128, 232, 292, 381  
 DDF 343  
 DDL 36, 119, 210, 224, 249, 335, 359, 363, 400, 414, 422  
 DECFLOAT 188  
 decimal 23, 168, 175  
 degree of parallelism 90, 144–145, 170, 172, 207, 312, 376  
 DELAY 212, 383  
 DELETE 6, 40, 65, 69, 120, 143, 171, 208, 245, 277, 313–314, 327, 398, 430–431  
 delete 9, 33, 35, 49, 69, 119, 174, 218, 237, 249, 268, 298–299, 321, 380, 398, 424, 430  
 DFSMS xxiii, 14, 22, 28–29, 147, 160, 180, 240, 325, 339–340, 378–379, 418  
 DFSMSdss 243, 246, 268, 323  
 DFSMSshm 5, 81, 252, 340, 405  
 DFSORT 15, 20, 90, 141–142, 311, 386  
 debugging 156  
 main memory 151  
 DFSORT default options 148  
 DFSORT installation parameter

DSA 149  
 DYNALOC 149  
 EXPMAX 149  
 EXPOLD 149  
 HIPRMAX 149  
 MAXLIM 149  
 MOSIZE 149  
 SIZE 149  
 SMF 150  
 TMAXLIM 149  
 DIAGNOS 55  
 DISCARD 10, 18, 77, 175, 205  
 DISPLAY UTIL 230  
 DISPLAY UTILITY command 5, 276, 383–384  
 distributed DB2 systems 414  
 distribution statistics 294  
 DPSI 15, 30, 144, 267  
 DRAIN 22, 209, 212, 324, 396, 430–431  
 DRAIN\_WAIT 208–209, 324, 430  
 DRDA 8, 20, 193, 195  
 DS8000 378  
 DSN1CHKR 13  
 DSN1COMP 13, 16, 37, 242, 418  
 DSN1COPY 13, 37, 77–79, 124–125, 161, 168, 177, 223, 226, 247, 400, 418, 466  
 DSN1COPY RESET 23, 79  
 DSN1LOGP 14, 16  
 DSN1LOGP utility 14  
 DSN1PRNT 14–15, 77–79, 224, 418  
 DSN1SDMP 15  
 DSN6SPRM 402  
 DSN6SYSP 274, 402, 407  
 DSNACCOR 135, 302, 448  
 DSNACCOX 15, 23, 302, 395  
 DSNDB06 117–118, 238, 292  
 DSNHDECP 188  
 DSNJCNVB 12  
 DSNJLOGF 12  
 DSNJU003 11–12, 370  
 DSNJU004 13, 347, 351  
 DSNRTSDB 126  
 DSNTIP6 252, 355  
 DSNTIP7 181  
 DSNTIPB 44, 403  
 DSNTIPL 274  
 DSNTIPO 311  
 DSNU1034I 66  
 DSNU116I 23, 276–277  
 DSNU1553I 255  
 DSNU364I 91, 173, 386  
 DSNU397I 91, 170, 172–173, 207, 209, 228, 264, 386, 431  
 DSNU427I 91, 228–229, 264, 387  
 DSNUM 71, 114–115, 121, 225, 261, 297, 356–358, 443, 445  
 DSNUTILB 127, 154, 346, 385, 429  
 DSNUTILS 5, 73, 154–155, 202–203, 245  
 DSNZPARM 21, 33, 95, 257, 293, 351, 355, 378  
 DSSIZE 18, 28–29, 33, 35, 378  
 DSTATS 294



dump classes 348  
dumpclass 341  
dynamic utility jobs 15, 21

## E

EBCDIC 61, 91, 171–172, 234–235, 325, 349, 356, 396, 430  
EDITPROC 23, 176  
efficiency 147, 265, 331  
element 48, 317  
ENDEXEC 127, 195, 197, 210, 430  
environment 5, 13, 99–100, 202–203, 255, 341, 388, 457  
ESS 403, 440  
ESTABLISH FCINCREMENTAL 352  
EVENT 55, 73, 227, 279, 355  
EXCLUDE 49, 186, 260, 445  
EXEC SQL 7, 19, 127, 195, 390, 430  
EXISTS 128  
EXPLAIN 294  
Explain 414, 461  
expression 48  
EXTENTS 114–115, 297, 345, 445

## F

Family Cross Loader 17  
FARINDREF 99, 292, 446  
FAROFFPOS 97–98  
FETCH 124–125  
file reference variables 189  
FILEDATA 65  
FILSZ 146  
FlashCopy 16, 18, 252, 323  
flexibility 22, 55, 60, 252, 436, 443  
FMID 430  
FREEPAGE 109–110, 292  
full copy 235  
FULL NO 235  
FULL YES 235  
function 6, 29, 60, 94, 138, 181, 187, 215, 224, 252, 291, 324, 341, 348, 395, 422

## G

GENERATED ALWAYS 194  
GENERATED BY DEFAULT 194  
GRAPHIC 182  
GRECP 274

## H

handle 9, 20, 188, 224, 331, 355, 407  
HFS 18, 22, 64, 82, 167  
High Performance Unload 168  
HISTOGRAM 18, 291, 310  
Histogram statistics 299  
HISTORY ALL 116, 296, 311, 316  
HPGRBRBA 272

## I

I14047 20  
IBM InfoSphere Classic Federation Server 195  
ICE046A 147, 156, 380  
ICTYPE 58, 77, 225, 361, 443, 446  
IDCAMS 64  
IFCID  
    3 213  
IGNOREFIELDS YES 196  
IGNSORTN 21, 160, 312, 402  
I113495 20, 150, 465  
I114047 465  
I114213 465  
I114296 21, 159  
Image Copy 59, 439–440  
IMPLICIT 385, 405  
IMS xxiii, 288, 407  
INCLUDE 49, 174, 186, 229, 235, 304, 362, 452  
incremental copy 235  
incremental FlashCopy 18, 22, 253, 348, 351  
INCURSOR 127, 195, 197  
index 5–6, 28–29, 48, 76, 93, 110, 141, 170–171, 205, 224, 251, 253, 293, 321, 364, 380, 415, 417  
Index recovery from Copy 277  
index usage tracking 125  
INDEXSPACESTATS 161, 302, 445  
Infinity 188  
InfoSphere Replication Server 18  
inline copy 41, 76, 78, 120, 170–171, 207–208, 224, 355  
INSERT 40, 120, 198, 336  
insert 107, 127, 184, 198, 237, 335, 364, 397, 424, 430  
INSERT processing 121, 198  
installation 147, 157, 181–182, 252, 311, 390  
INSTANCE 118, 337  
IRLM 384  
IS 58, 61, 184–185, 216, 240, 247, 302, 325, 352, 357, 391, 433  
ITEMERROR 55, 69, 227, 279, 355  
IX 112, 422, 430

## J

JDB991K 19  
JOBNAME 58, 61, 153, 286, 385, 391

## K

KEEPDICTIONARY 60, 175–176, 212, 247, 299, 396, 402  
keyword 7, 10, 29, 48, 61, 75, 79, 109, 111, 143, 174, 207, 218, 224, 235, 294–295, 330, 403, 405, 425, 427

## L

LARGE 28, 38, 160, 252  
LASTUSED 125  
LEAFDIST 110, 316, 447  
LEAFDISTLIMIT 10, 110, 301  
LEAFFAR 111, 316, 447  
LEAFNEAR 107, 316, 447  
LEAST 294–295

let 155, 171–172, 208–209, 266  
 LIKE 127, 423  
 list 7, 35, 48, 76, 80, 96, 119, 168, 227, 233, 253, 276,  
 293, 303, 321, 343–344, 383, 402, 415–416, 465  
 LISTDEF 8, 35, 48–49, 74, 85–86, 88, 174, 186, 227,  
 233, 235, 294, 303, 355, 361–362, 390, 452, 466  
 LISTDEF expansion 51, 72, 261  
 LOAD 7–8, 31, 55, 60, 64–65, 76, 105, 116, 141, 143,  
 167, 206, 208, 268, 291, 299, 301, 378–379, 385,  
 425–426  
 LOAD partition parallelism 23  
 LOB 5–6, 36, 52, 78, 124, 167–168, 205, 247, 291, 297,  
 321, 380, 417, 420, 467  
 LOB column 7, 193–194, 316, 330  
 LOB data 5, 168, 189, 217, 247, 319, 331  
 LOB table 6, 132, 181, 214, 258, 316, 321  
 LOB table space 6, 190, 217, 317, 321  
 LOBs 6, 22, 36, 133, 190, 218–219, 316, 321, 425  
     allocation 219  
     processing 22  
     update 219, 332  
 LOCATE 18, 22, 323, 399  
 locking 38, 423  
 locks 10, 38, 187, 199, 394, 397  
 LOG 22, 41, 65, 77, 119, 170, 172, 207, 216, 224, 252,  
 330, 342, 396, 401, 431  
 Log Apply 76, 79, 265, 273  
 LOG NO 77, 127, 171, 183, 207, 225, 252, 330, 432  
 Log records sorting 275  
 LOGAPPLY 11, 23, 254, 276  
 LOGGED 39, 168, 185, 218, 322  
 logging 40, 76, 186, 218–219, 262, 344, 400  
 LOGICAL\_PART 40, 43  
 LOGONLY 262, 269, 408, 457  
 LONGLOG 213, 383  
 LPL 18, 186, 274  
 LRSN 9, 11, 79–80, 103, 210, 227, 253, 277, 347, 357,  
 399, 430

## M

maintenance 20, 54, 136, 175, 212–213, 227, 375, 392,  
 423, 465  
 map page 38, 218, 233  
 mass delete 38  
 MAXERR 182  
 MAXRO 209, 212, 383, 430–431  
 MERGE 70, 87, 142, 386  
 MERGECOPY 7, 50, 60, 224, 390, 459  
 MGEXTSZ 181, 378  
 MGMTCLAS 325, 378  
 MODIFY 8, 50, 214, 226, 252, 291, 298, 361, 390, 398,  
 417  
 Modify 12, 17, 281–282, 313, 398, 417, 420  
 MODIFY RECOVER 313  
 MODIFY STATISTICS 9, 23, 50, 296, 299  
 MQT 185, 199, 294

## N

NaN 188

NEARINDREF 99, 292, 446  
 NEAROFFPOS 98, 109  
 NEAROFFPOSF 97  
 NFM 23, 120, 126, 176, 252  
 node 219, 240, 270, 330, 422  
 NOSYSREC 85, 207–208, 396  
 NOT LOGGED 185, 262, 330  
 NOT LOGGED table spaces 185–186, 262, 330  
 NOT PADDED 106, 259  
 Note 22, 34, 53, 75–76, 100, 111, 150, 174, 229, 255,  
 309, 341, 345, 378  
 NPAGESF 115, 286, 446  
 NPI 22, 29, 85, 174, 214, 225  
 NULL 21, 106, 119, 192, 210, 302, 335, 391, 430  
 NUMPARTS 29, 32

## O

OA23489 370  
 OA24814 370  
 OA25204 64  
 OA25206 64  
 OA25280 64  
 Object 48, 80, 138, 336, 339, 355, 435–436  
 OFFPOS 100, 103  
 OFFPOSLIMIT 42, 98, 301  
 OFFPOSLIMIT 98  
 Online Schema 17–18, 395  
 online utilities 5, 66, 321, 375, 387, 429  
 Optim Query Tuner 460  
 Optim Query Workload Tuner 460  
 optimization 301  
 Optimization Service Center 117, 461  
 OPTION PREVIEW 54  
 OPTIONS 9, 48, 72, 151, 227, 260, 311, 390  
 options 9, 37, 55, 77, 79, 107, 110, 168, 170, 205–206,  
 223, 252, 298, 301, 322, 348, 387, 393, 415  
 OPTIONS(PREVIEW) 72  
 OPTIOWGT 466  
 OPTIXOPREF 466  
 ORDER 127–128, 196, 245, 293, 382  
 ORDER BY 128–129, 197, 382

## P

packaging 19  
 PADDED 106, 259  
 page size 105, 241, 271, 394  
 PAGESAVE 107, 115, 135, 299, 446  
 Parallel Access Volume 175, 212  
 Parallel Index Build 75–76, 143–144, 170, 207  
 PART 29, 58, 77, 111, 172, 209, 301, 303, 337, 381, 431  
 PARTITION 30, 118, 176–177, 268, 301–302, 382  
 partition by growth 199  
 Partition Parallelism 17–18, 76  
 partition-by-growth table space 32  
 partition-by-growth table spaces 33, 85, 179  
 PARTITIONED 31  
 partitioned 29  
 partitioned table space 10, 14, 28, 51, 76, 100, 168, 170,  
 205, 207, 267, 381, 393, 437

- partitioning xxiii, 18, 27–29, 86–87, 174, 215, 229, 279, 290, 301, 381, 393
  - impact on SQL 43
- partitions 4, 9, 13, 27, 33, 67, 76, 100, 111, 170, 205, 231, 233, 264, 292, 301, 381, 389, 395, 437, 443
  - altering 43
- PARTKEYU 44–45
- PARTLEVEL 35, 53, 85, 88, 174, 229, 260, 303, 452
- PATHDISP 65
- PATHOPTS 65
- PBG 33, 176–177
- PBR 37
- PCTFREE 100, 109, 292
- PERCDROP 112, 115, 446
- performance xxiii, 4, 28, 38, 67, 75, 80, 95, 108, 141, 168, 170, 205, 262, 268, 291–293, 363, 375, 377, 401, 414, 423, 465
- Performance Query 395, 414
- PGFIX 394
- PIB 75–76, 143–144, 170–171, 207–208
- pieces 18, 29
- PIECESIZE 29, 447
- pipe 64
- PK01155 152
- PK25047 156
- PK26989 90, 386, 407–408
- PK34251 202
- PK41182 465
- PK41711 326, 370, 405, 465
- PK41899 21, 159, 162, 465
- PK42573 104, 465
- PK42768 465
- PK42768. 104
- PK45916 21, 159, 162
- PK47083 175, 212, 401, 465
- PK51853 180, 402, 466
- PK53341 161
- PK55972 328, 334, 466
- PK60612 187, 466
- PK60956 175, 212, 466
- PK60956, 175
- PK61759 175, 212, 466
- PK63324 397, 466
- PK63325 397, 466
- PK63409 156
- PK66899 156
- PK67154 212, 466
- PK70269 64, 200, 203, 466
- PK70866 175, 212, 466
- PK74993 466
- PK75216 175, 193, 466
- PK75463 466
- PK75643 293
- PK76860 175, 466
- PK77313 175, 466
- PK77426 293, 466
- PK78516 466
- PK78865 466
- PK78958 176
- PK789858 466

- PK79136 152, 175, 212, 467
- PK79144 400
- PK81232 467
- PK83096 467
- PK83996 385, 467
- PK85856 21, 155, 467
- PK85881 33, 467
- PK85889 21, 155, 175, 212, 467
- PK87348 176, 467
- PK87762 467
- PK89645 373, 467
- PK89889 156, 467
- PK92248 153, 467
- PK92725 467
- PK97713 142, 467
- PM03691 178, 467
- POSITION 171, 432
- PQ50666 314
- precision 188
- predicate 300
- prefix 310, 382, 422
- PREVIEW 54, 66, 72, 260
- PRIQTY 39, 112, 175, 181, 210, 218, 378, 430
- production environment 376
- PSEUDO\_DEL\_ENTRIES 112, 114, 316, 447
- PSRBD 37

## Q

- QMF 407–408
- quantiles 23, 299
- query 10, 28–29, 93, 167, 195, 215, 292, 298, 381–382, 424
- query optimization 23
- query performance 100, 302, 306, 461
- QUIESCE 9, 49–50, 118, 226, 270, 288, 390, 398, 439

## R

- RACF 183, 241, 325, 379
- range-partitioned table space 32, 37
- RBA 9, 12, 60, 79–80, 174, 227, 253, 277, 317, 357, 399, 457
- Real Time Statistics 15
- real-time statistics 15, 117, 160, 294, 423, 425
- REBALANCE 18, 36, 41, 85
- REBUILD 50
- REBUILD INDEX 9, 19–20, 31, 36, 51, 76, 79, 119, 121, 129, 141, 159, 225, 260, 291, 299–300, 323, 358, 361, 378, 405, 427–428
- rebuild pending 330, 364
- RECOVER 7, 36, 50, 76–77, 121, 185, 219, 224, 251, 300, 313, 334, 355, 382, 384, 439
  - from CONCURRENT COPY 271
  - without an image copy 272
- RECOVER DSNUM 31
- RECOVER PARALLEL 50, 81, 263, 267
- Recovery Expert 139
- recovery pending 77
- RECOVERYDDN 56, 77, 227
- RECP 186, 218, 330, 458

Redbooks Web site 470  
 Contact us xxvi  
 REGION 90, 234, 346, 376–377, 449, 451  
 region sizes 154  
 reordered row format 168, 176, 215, 259, 402  
 REORG 7–8, 10, 31, 50, 60, 76, 95, 111, 141, 143, 168,  
 176–177, 205, 247–248, 291–292, 355, 360, 378–379,  
 415  
     BUILD2 215  
     DRAIN 213, 430  
     KEEPDICTIONARY 181, 402  
     LOG phase 78, 215  
     MAXRO 213, 430  
     REUSE 112, 206, 396  
     SHRLEVEL CHANGE 22, 41, 79, 198, 206, 271, 381,  
     430  
     SHRLEVEL REFERENCE 10, 41, 43, 77–78, 236,  
     380  
     SORTDATA 60, 208, 396  
     SORTKEYS 23  
     SWITCH 215  
     TIMEOUT 212–213, 431  
     UNLOAD EXTERNAL 10  
 REORG TABLESPACE 10, 36, 76, 95, 176, 205, 215,  
 225, 299, 379, 383, 415, 430  
 REORP 41, 43  
 REPAIR 10, 55, 226, 248, 321, 390, 394, 439  
 REPAIR VERSIONS 400  
 REPORT 11, 50, 91, 116, 226, 254, 303, 332, 390, 401  
 REPORT RECOVERY 11, 268, 284  
 REPORT TABLESPACESET 11, 262  
 REPORT utility 11, 285  
 REPORTONLY 10, 42, 104, 111, 231, 418  
 requirements 21, 48, 90, 183, 219, 236, 298, 379, 386  
 RESET 23, 40, 79, 247  
 resource unavailable 18, 22, 33, 213, 328, 396  
 RESTART 15, 23, 71, 122–123, 143, 189, 194, 260, 385  
 restartability 69, 189  
 RESTORE xxiii, 5, 11, 82, 243, 252, 339, 342, 370, 403,  
 457  
 RESTORE SYSTEM 11, 252, 351, 370, 403, 457  
 RESTRICT 40, 136, 226  
 restricted status 69  
 RESUME YES 65, 116, 127, 179–180, 206, 432  
 RETRY 208–209, 324, 396, 405, 430  
 RETRY\_DELAY 208–209, 324, 430  
 return 6, 9, 49, 64, 186, 212, 227, 233, 303, 332  
 REUSE 174, 212, 243  
 RID 96, 216, 333, 382  
 RIDs 96, 210, 382  
 ROLLBACK 122  
 ROTATE PARTITIONS 40  
 row format 168, 176, 215, 259, 402  
 ROWID 194, 331  
 RRSF 407–408  
 RTS 93, 294, 389, 395, 423–424  
 RUNSTATS 11, 31, 41, 50, 55, 76, 107, 110, 141, 143,  
 170–171, 173, 207, 221, 291–292, 383, 390, 395, 423  
 RUNSTATS option  
     KEYCARD 304

SAMPLE 306  
 RUNSTATS TABLESPACE 50  
 RVA 403

## S

SAP 282, 369, 378  
 SBSCS 182, 220, 335  
 schema 199  
 SCOPE PENDING 16, 40–41, 233–234  
 SDSNEXIT 127, 247, 346, 352, 422, 451  
 SECQTY 39, 175, 181, 210, 218, 378, 430  
 SECURITY 183  
 segmented table space 32–33, 131–132, 171, 208, 231  
 SEQ 58, 286, 368, 403  
 SEQCACH 403  
 sequential files 169  
 SET 120, 226, 252, 325, 360, 391  
 shadow data sets 16, 86, 206, 212, 323, 380  
 SHRLEVEL 7, 18, 35, 56, 76–77, 116, 168, 175, 206,  
 227, 254, 301, 317, 322, 370, 378, 380, 418, 430  
 SHRLEVEL CHANGE 323  
 side 52, 380  
 signaling NaN 188  
 simple table space 33, 112  
 SKIP LOCKED DATA 23, 199, 397  
 SMF 142  
 SMF record type 16 150  
 SORTBLD 82–83, 172, 208, 361, 383, 386, 431  
 SORTDATA 18, 212  
 SORTDEVT 85, 143–144, 171–172, 207, 311, 324, 430  
 SORTKEYS 17–18, 23, 75, 82, 168, 170–171, 432  
 SORTNUM 15, 21, 90, 145, 170–171, 207–208, 312,  
 377, 384  
     elimination 387  
 SORTNUM elimination 91, 159  
 SORTWKnn data sets 143  
 SPACE 56, 61, 91, 107, 110, 146, 171, 208, 229, 234,  
 269, 296, 301, 325, 345, 384–385, 423  
 SPACEF 113–114, 297, 446  
 SQL 7, 27–28, 97, 113, 167, 169, 210, 220, 237, 267,  
 294, 332, 385, 387, 414  
 SQL statement 125, 140, 195, 312, 337, 397, 461  
 SQLCODE 38, 328, 430  
 SSID 58, 415  
 STACK 56, 66, 415, 452  
 stand-alone utilities 5, 12, 78, 242, 375  
 START DATABASE 118, 125, 226, 274  
 STATCLUS 95–96, 293, 404  
 statement 5, 7, 29, 48, 85–86, 113, 125, 172, 174, 211,  
 214, 227, 258, 292, 294, 337, 347, 397, 401, 414, 416  
 statistics 4, 6, 60, 76, 93, 170, 191, 207, 263, 291, 386,  
 389, 413–414  
 Statistics Advisor 140, 294  
 statistics enhancements 23  
 STATISTICS option 120, 301  
 STATSINT 118, 121, 161  
 STATUS 230, 234, 337, 383, 446  
 STORCLAS 325, 378, 405  
 STOSPACE 11, 55, 390  
 STYPE 77, 240

substring notation 58  
 SYSCOLDIST\_HIST 296  
 SYSCOLUMNS 221, 295, 303, 431, 447  
 SYSCOLUMNS\_HIST 296  
 SYSCOPY 8–9, 40–41, 70, 77–78, 80, 135, 177, 188,  
 214, 219, 225, 233, 253, 355, 396, 398, 446, 459  
 SYSDBASE 97, 292, 296  
 SYSHIST 134, 296, 316  
 SYSIBM.SYSCOLUMNS\_HIST 296, 313  
 SYSIBM.SYSCOPY 9, 43, 80, 177, 239, 281, 459  
 SYSIBM.SYSINDEXES 95, 277, 318, 381  
 SYSIBM.SYSINDEXSPACESTATS 21, 117, 122, 124  
 SYSIBM.SYSTABLEPART 30–31, 43, 128, 177, 292,  
 381–382  
 SYSIBM.SYSTABLES 317  
 SYSIBM.SYSTABLESPACE 34, 128, 359  
 SYSIBM.SYSTABLESPACESTATS 21, 117, 122, 124  
 SYSIN 9, 48–49, 74, 91, 127, 211, 234, 304, 325, 346,  
 349, 393, 396, 430  
 SYSINDEXES 30–31, 95, 109, 221, 277, 318, 381, 447  
 SYSINDEXES\_HIST 95, 105, 296  
 SYSINDEXPART 30, 41, 110–111, 221, 271, 292, 316,  
 381–382, 431, 447  
 SYSINDEXPART\_HIST 105, 110, 296  
 SYSINDEXSPACESTATS.LASTUSED 123  
 SYSLGRNX 8–9, 41, 119, 225, 357, 359, 385, 398  
 SYSLOBSTATS 132, 317  
 SYSLOBSTATS\_HIST 296  
 SYSPITR 11, 370  
 SYSPRINT 72, 127, 241, 247, 303, 345–346, 392, 432,  
 451  
 SYSTABLEPART 30–31, 99, 103, 105, 177, 221, 271,  
 292, 381–382, 431, 446  
 SYSTABLEPART\_HIST 105, 115, 296  
 SYSTABLES 105, 115, 221, 317, 431, 446  
 SYSTABLES\_HIST 105, 115, 296  
 SYSTABSTATS\_HIST 296

## T

table space scans 67, 214  
 tables 6, 29–30, 48, 94, 96, 167, 206, 235, 248, 259,  
 291–292, 321, 340, 382, 402, 423  
 TABLESPACESTATS 128, 161, 302, 445  
 TCP/IP 202  
 TEMPLATE 9, 55–56, 61, 83, 143, 171, 208, 229, 233,  
 260, 328, 390, 415  
 template switching 60  
 templates 4, 11, 47, 54–55, 232, 448–449
 

- benefits 55
- how to use 60
- naming standards 57, 68
- recommendations 66
- restrictions 66

 templates and partitions 71  
 TIME 56, 61, 121, 172, 208, 229, 234, 277, 325, 346,  
 384, 404, 431  
 TIMESTAMP 127, 171, 173, 221, 238, 286, 335, 432  
 TOTALROWS 21, 119–120, 302, 445  
 triggers 36, 40, 199, 335  
 TS 18, 51, 61, 111, 229, 234, 277, 307, 396, 422, 438

TYPE 34, 38, 210, 286, 300, 337, 344, 358, 396, 430

## U

UA28767 351  
 UA42371 370  
 UA42372 370  
 UDFs 196  
 UK24255 465  
 UK25468 465  
 UK25469 465  
 UK28089 351  
 UK31489 466  
 UK33636 21, 159, 465  
 UK33692 21, 159  
 UK34198 465  
 UK34808 466  
 UK35132 187, 466  
 UK36306 466  
 UK37143 466  
 UK37144 466  
 UK39612 466  
 UK41370 370, 465  
 UK42565 293, 466  
 UK42884 466  
 UK43355 466  
 UK43512 466  
 UK43948 64, 466  
 UK43977 466  
 UK44703 467  
 UK45138 466  
 UK45192 467  
 UK45353 466  
 UK45791 466  
 UK45998 466  
 UK46236 466  
 UK47616 467  
 UK48846 21, 467  
 UK48911 21  
 UK48912 21, 467  
 UK49339 467  
 UK50412 467  
 UK50413 467  
 UK50430 467  
 UK51104 467  
 UK51283 466  
 UK51396 467  
 UNICODE 74  
 Unicode 179  
 UNIQUE 194, 210, 335, 391, 430  
 universal table space 32, 85, 199, 335  
 UNLOAD 7, 37, 42, 50, 64, 66, 76, 78, 105, 139, 167,  
 208–209, 301, 325, 358, 361, 386, 415
 

- FROM TABLE 186, 200
- output data sets 174
- phases 214, 386

 UNLOAD EXTERNAL 12, 42, 168  
 unloading from a table space 199  
 UPDATE 43, 110, 146, 173, 221, 296, 298, 401–402,  
 431  
 UQ56653 314

UTF-8 74, 179  
UTIL\_TEMP\_STORCLAS 370  
utilities  
    changes across versions 15  
UTRO 37, 213, 243, 290, 332, 396  
UTRW 18, 22, 227  
UTS 27, 32, 177, 199  
UTSORTAL 312  
UTSORTAL=NO 146  
UTSORTAL=YES 91, 160, 406  
UTUT 37, 213, 267, 332

## V

VALIDPROC 176  
VALUES 29–30, 243, 305, 310, 336, 391  
VARCHAR 38, 106, 122, 146, 182, 220, 335, 363, 432  
VARGRAPHIC 182  
variable 20, 58, 168, 172, 274  
VERSION 243, 333, 346  
Version xxiii, 4–5, 28, 49, 81, 117, 174, 179, 214, 252,  
299, 323, 339, 351, 392, 430  
versions 15, 18, 31, 86, 282, 303, 340, 347, 400  
VIO 147  
Virtual I/O 147

## W

WHEN option 187, 196  
Wildcarding  
    benefits 48  
wildcarding 47  
    benefits 48  
wildcards 48, 50, 260  
    restrictions 55  
wildcards and templates 66  
WITH 127, 213, 216, 234, 325, 349, 355, 391, 396  
WLM 155, 384  
workfile 387  
write-down 183

## X

XML 4, 6, 37–38, 78, 129, 167–168, 205, 302–303, 321  
    columns 38, 40, 42, 79, 191, 219–220, 262–263, 302,  
    319, 330  
    data type 5, 219  
    documents 190, 217, 319, 331

## Z

z/OS xxiii, 5, 57, 81, 176, 214, 252, 342, 377  
zIIP 4, 15, 90, 141, 386



**Redbooks**

# DB2 9 for z/OS: Using the Utilities Suite

(1.0" spine)

0.875" <-> 1.498"

460 <-> 788 pages









# DB2 9 for z/OS: Using the Utilities Suite

## Description of the utilities and recent enhancements

## Recommendations for best performance and availability

## Advice for operational scenarios

IBM continues to enhance the functionality, performance, availability, and ease of use of IBM DB2 utilities.

This IBM Redbooks publication is the result of a project dedicated to the current DB2 Version 9 Utilities Suite product. It provides information about introducing the functions that help set up and invoke the utilities in operational scenarios, shows how to optimize concurrent execution of utilities and collect information for triggering utilities execution, and provides considerations about partitioning.

It also describes the new functions provided by several utilities for SHARE LEVEL CHANGE execution, which maximize availability and the exploitation of DFSMS constructs by the BACKUP and RESTORE SYSTEM utilities.

This book concentrates on the enhancements provided by DB2 UDB for z/OS Version 8 and DB2 for z/OS Version 9. It implicitly assumes a basic level of familiarity with the utilities provided by DB2 for z/OS and OS/390 Version 7.

## INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

## BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

**For more information:**  
[ibm.com/redbooks](http://ibm.com/redbooks)

SG24-6289-01

ISBN 0738434051